Sistemas Informáticos
Curso 05-06

MOVA Tool
(Visual ITP/OCL Tool)

Fernando Alcaraz Martín
Jorge Arias Baña
Juan Pablo Gavela Coya

Dirigido por:
Prof. Manuel García Clavel
Departamento de Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Cesión de derechos

Los alumnos abajo firmantes: Fernando Alcaraz Martín, Jorge Arias Baña y Juan Pablo Gavela Coya, autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y el sistema desarrollado: MOVA Tool (Visual ITP/OCL Tool).

Fernando Alcaraz Martín Jorge Arias Baña Juan Pablo Gavela Coya

Palabras clave para la búsqueda bibliográfica

UML
OCL
MOVA
ITP/OCL Tool
Maude
Diagrama de clases
Diagrama de objetos
Restricciones
Invariante
Validación

Resumen del proyecto

El objetivo de este proyecto es implementar un entorno gráfico de desarrollo de diagramas UML que permita la interactuación con el motor de validación ITP/OCL Tool. Dicho sistema, denominado MOVA Tool (Modelling & Validation Tool) es capaz de especificar, verificar y validar modelos UML+OCL aprovechando las características de ITP/OCL Tool.

Existen dos partes diferenciadas en MOVA: edición de diagramas y validación modelos. La edición de diagramas permite crear fácilmente diagramas de clases y de objetos usando las opciones gráficas que brinda la herramienta. Los diagramas de clases permiten crear clases, clases enumeradas, y distintos tipos de relaciones entre ellas. Los diagramas de objetos permiten instanciar clases y crear enlaces entre objetos creados.

La validación de modelos consiste en asociar invariantes escritos en OCL a diagramas de clases y validarlos sobre diagramas de objetos relacionados instanciados a partir de dicho diagrama de clases. La validación se realiza a través del envío de comandos a ITP/OCL Tool a través del intérprete Maude. Se pueden ejecutar consultas sobre diagramas de objetos.

MOVA es capaz de editar grupos de diagramas; guardar y cargar diagramas en formato XMI, XML, EPS y en una base de datos; guardar y cargar invariantes en formato texto; aplicar zoom a diagramas e imprimir diagramas. Está escrito íntegramente en Java y puede ejecutarse en distintas plataformas.

Project's summary

The main aim of this project is implementing a graphical developing environment for UML diagrams that allows user interacting with ITP/OCL Tool validation engine. This system, called MOVA Tool (Modelling & Validation Tool) is able to specify, verify and validate UML+OCL models taking advantage of ITP/OCL Tool's features.

There are two main parts in MOVA Tool: model's edition and model's validation. Model's edition allows user to create easily class diagrams and object diagrams through tool's graphical options. Class diagrams contain non enumerated classes, enumerated classes and different kinds of associations between them. Object diagrams allow instantiating classes and creating links between created objects.

Model's validation consists on associating invariants written in OCL to class diagrams and checking them over the object diagram obtained from former class diagram. Validation is carried out sending commands to ITP/OCL Tool through Maude interpreter. Queries can be executed over an object diagram.

MOVA can edit groups of diagrams; save and store diagrams in XMI, XML and EPS formats and in a database; save and store invariants on text files; zooming diagrams and printing diagrams. The code is written in Java and can be run on several operating systems.

Nota

En el desarrollo de este documento nos referiremos a la herramienta desarrollada como Visual ITP/OCL Tool. Este nombre fue el acordado al comenzar el proyecto y se mantuvo durante todo el proceso de desarrollo.

Al terminar el proyecto, y con vistas a la libre distribución de la herramienta, se produjo el cambio de nombre por motivos legales al haber más herramientas denominadas con Visual y OCL. El nombre definitivo fue MOVA Tool.

Por consiguiente hacemos notar al lector que Visual ITP/OCL Tool es lo mismo que MOVA Tool, y que en el presente documento el nombre del sistema desarrollado que debe aparecer mencionado es MOVA Tool.

VISUAL ITP/OCL TOOL

Tabla de contenidos

1 INTRODUCCIÓN

En este documento se presenta la herramienta Visual ITP/OCL Tool, una herramienta gráfica de edición de diagramas y restricciones que ayuda a la validación automática de diagramas UML con respecto a restricciones OCL. La implementación de Visual ITP/OCL Tool se apoya en la herramienta ITP/OCL Tool, un intérprete de expresiones OCL basado en reescritura, que utiliza la especificación ecuacional de los diagramas UML+OCL.

El propósito de Visual ITP/OCL Tool es proporcionar un entorno gráfico de desarrollo de diagramas, a la vez que hace de mediador entre el usuario y el ITP/OCL Tool. Con esto se sustituye el trabajo con comandos a través de terminal por la interacción con el interfaz, lo que evita al usuario conocer las sintaxis de base y agiliza la comunicación. Visual ITP/OCL Tool ofrece además características adicionales propias de las herramientas de modelado.

1.1 UML y OCL

Como en todas las disciplinas de estudio, es importante buscar una forma común de comunicación (un lenguaje técnico) de modo que todos los individuos puedan intercambiar información de manera estándar sin importar el idioma o la localización. En el contexto software existe una manera precisa y fiable de representar los programas: el lenguaje unificado de modelado.

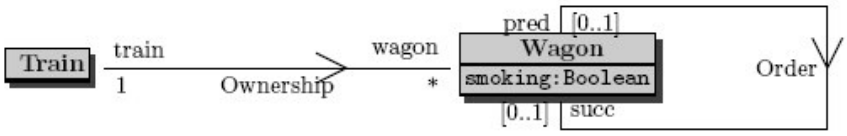
El lenguaje unificado de modelado o UML es un lenguaje de modelado visual de propósito general para especificar, visualizar, construir y documentar los distintos elementos de un sistema software. La notación utilizada se basa en diagramas de distintos tipos: casos de uso, clases, objetos, secuencia… Sin embargo, para ciertos aspectos de los modelos, estos diagramas no son tan concisos y expresivos como puede resultar un lenguaje textual. El lenguaje de restricciones sobre objetos OCL es un lenguaje de restricciones textual que complementa a los diagramas UML para proporcionar un mayor manejo del dominio de la representación. OCL ofrece ayuda a la hora de especificar información muy precisa sobre modelos UML.

Dentro de las fases de desarrollo de software, las fases de validación y prueba tienen sin duda una importancia clave. Hay muchas maneras de validar un modelo: simulación, prototipado rápido… Con OCL se validan los modelos comprobando si sus instancias cumplen las restricciones deseadas. Existen numerosas herramientas CASE (Computer Aided Software Engineering) que facilitan el diseño y la documentación con diagramas UML, como Rational Rose, Software Modeler o Poseidon. Sin embargo, hay poco soporte a la hora de validar modelos durante la fase de diseño, y por lo general ningún soporte específico para restricciones escritas en OCL, exceptuando USE Tool.

1.2 Diagramas UML + OCL

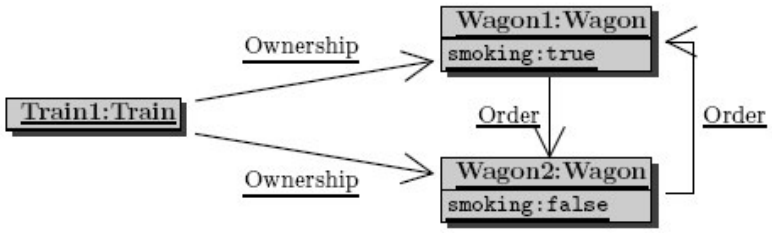
La vista estática de UML modela conceptos en el dominio de la aplicación así como conceptos internos diseñados como parte de la implementación de la aplicación. Dicha vista no explica el comportamiento de dependencia en el tiempo del sistema, el cual se explica en otras vistas. Elementos clave en la vista estática son las clases y sus relaciones, que pueden ser de diferentes tipos incluyendo asociaciones y generalizaciones. La vista estática se muestra en los diagramas de clase.

Ejemplo 1. Consideremos el diagrama de clases TRAINWAGON como muestra la siguiente figura que modela un ejemplo para un contexto de trenes. Un tren puede tener vagones, y estos vagones pueden estar conectados a otros vagones (vagones predecesor y sucesor). Los vagones pueden ser a su vez de fumadores y no fumadores.



Un sistema puede pasar por distintos estados a medida que varía con el tiempo. Los diagramas de objetos modelan los objetos y enlaces que conforman el estado del sistema en un instante concreto (forman una instantánea del sistema). Un objeto es una instancia de una clase y un enlace es una instancia de una relación. Los diagramas de objetos son una herramienta básica para la investigación y la experimentación.

Ejemplo 2. Consideremos ahora el diagrama de objetos TRAINWAGON-1 que muestra la siguiente figura. Este diagrama describe una instantánea del sistema de trenes modelado por el diagrama de clases TRAINWAGON anterior. Dicha instantánea puede considerarse no deseada puesto que describe un tren con dos vagones unidos entre sí de manera cíclica.



OCL es un lenguaje de especificación puro que actúa por encima de UML. Es un lenguaje textual con un estilo de notación similar al de los lenguajes orientados a objetos comunes. La expresión de restricciones se hace a través de invariantes, que no son más que cadenas que expresan una propiedad a cumplir.

Ejemplo 3. Consideremos la siguiente restricción sobre el diagrama de clases TRAIN-WAGON: "No existen dos vagones diferentes directamente enlazados entre sí de manera cíclica". Esta restricción se puede expresar usando OCL con el siguiente invariante notInCyclicWay sobre el diagrama de clases TRAINWAGON:

```
not ((Wagon.allInstances)
→ exists(w1:Wagon | (Wagon.allInstances) → exists(w2:Wagon |
w1:Wagon <> w2:Wagon
and (w1:Wagon.succ) → includes(w2:Wagon)
and (w2:Wagon.succ) → includes(w1:Wagon)))) .
```

El diagrama de objetos TRAINWAGON-1 no satisface esta restricción puesto que hay dos vagones, llamados Wagon1 y Wagon2, dispuestos de modo que los sucesores de Wagon1 incluyen a Wagon2 y viceversa. Por tanto el chequeo del invariante será fallido.

1.3 ITP/OCL Tool

La herramienta ITP/OCL Tool se basa directamente en especificaciones ecuacionales de los diagramas UML+OCL. En concreto:

- 1) Los diagramas de clases y de objetos UML están especificados como teorías ecuacionales de pertenencia.
- 2) Las restricciones OCL están expresadas como términos booleanos sobre extensiones de las teorías de 1).
- 3) La validación de diagramas de objetos con respecto a las restricciones se reduce a comprobar si los términos booleanos correspondientes se reescriben a cierto o falso.

ITP/OCL Tool está escrito completamente en Maude, un lenguaje de programación basado en la reescritura de términos que implementa la lógica ecuacional de pertenencia y la lógica de reescritura. Maude es un lenguaje de programación reflexivo. Esto significa que tanto el analizador sintáctico como el motor de reescritura están disponibles para el programador como operaciones: de hecho se utiliza lo primero para implementar el analizador sintáctico OCL en la herramienta, y lo segundo para implementar el motor de validación UML+OCL. El tamaño del código de esta herramienta ronda 4.000 líneas.

La implementación de una herramienta interactiva en Maude comprende cuatro tareas principales:

- Definir un bucle continuo lectura-evaluación-impresión.
- Definir la sintaxis de las órdenes.
- Definir la interacción con el bucle continuo.
- Definir el procesamiento de las órdenes.

Maude ofrece facilidades para entrada/salida genérica a través del uso de sus "*loop objects*". Proporciona además gran flexibilidad para definir la sintaxis de las órdenes gracias a su terminal y al uso de "*bubbles*" (cualquier lista no vacía de identificadores Maude). Finalmente, el procesamiento de peticiones en una herramienta interactiva viene definido en Maude por ecuaciones actuando sobre los *"loop objects"*.

Las órdenes de la herramienta ITP/OCL puede agruparse en cuatro tipos:

- *Órdenes de creación de diagramas*: Ecuaciones de inserción de diagramas, ya sean de clases o de objetos.
- *Órdenes de inserción de elementos (clase, atributo, relación…) en un diagrama:* añaden clases, atributos, relaciones, generalizaciones y asociaciones a un diagrama de clases, y objetos, valores y enlaces a un diagrama de objetos.
- *Órdenes que fijan una restricción a un diagrama de clases:* asocian restricciones a diagramas de clases por medio de invariantes definidos por el usuario.
- *Órdenes que validan un diagrama de objetos:* chequean las restricciones asociadas al diagrama de clases sobre los objetos del diagrama de objetos asociado.

Visual ITP/OCL Tool transforma las acciones del usuario en comandos de los grupos anteriormente citados. Estos comandos se ejecutan a través del intérprete Maude y devuelven el resultado.

Para más información sobre ITP/OCL Tool visite: http://maude.sip.ucm.es/itp/ocl/.

2 MANUAL DE USUARIO

2.1 Consideraciones previas a la ejecución

Consideraremos que el usuario tiene un archivo ejecutable de Visual ITP/OCL Tool compilado para su sistema, o en su defecto un archivo ejecutable de Java con extensión *jar*. Este ejecutable puede ubicarse en cualquier carpeta que el usuario desee. Para el correcto funcionamiento del programa se debe incluir una carpeta adicional de componentes visuales para iconos. Esta carpeta debe llamarse obligatoriamente *pics* y debe estar en el mismo directorio que el ejecutable de Visual ITP/OCL Tool. La carpeta *pics* se incluye en esta distribución.

Para poder ejecutar la herramienta necesitaremos dos elementos auxiliares:

- *ITP/OCL Tool*: verificador de restricciones básico para Visual ITP/OCL Tool. Disponible de forma gratuita en: http://maude.sip.ucm.es/itp/ocl/. Se recomienda descargar la última versión disponible.
- *Maude*: intérprete necesario para ejecutar ITP/OCL Tool. Disponible de forma gratuita en http://maude.cs.uiuc.edu/.

En el mismo directorio del archivo ejecutable se crearán dos archivos más tras la primera ejecución. Estos archivos son: *preferencias.txt* y *log.txt*. El archivo de preferencias guarda las rutas necesarias para ejecutar la herramienta, mientras que el archivo de log permite consultar los comandos que se envían a ITP/OCL Tool. Se recomienda no manipular dichos archivos.

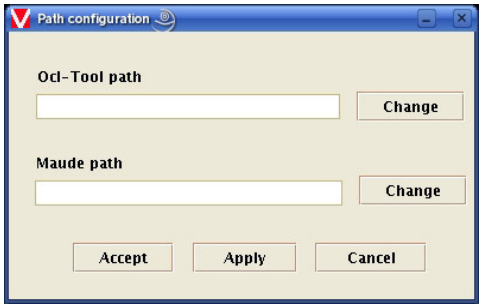
Una vez tenemos instalados Maude, ITP/OCL Tool y la carpeta *pics* podemos lanzar la herramienta.

NOTA: Hay que advertir que el usuario debe tener permiso de escritura en el directorio donde se encuentra el ejecutable.

NOTA: Si el usuario lo desea, puede utilizar una base de datos en MySQL para almacenamiento persistente de diagramas. Para ello, necesita el programa MySQL y realizar la ejecución de los comandos de creación de tablas. Estos comandos se incluyen en la distribución de Visual ITP/OCL Tool. El uso de la base de datos no es obligatorio.

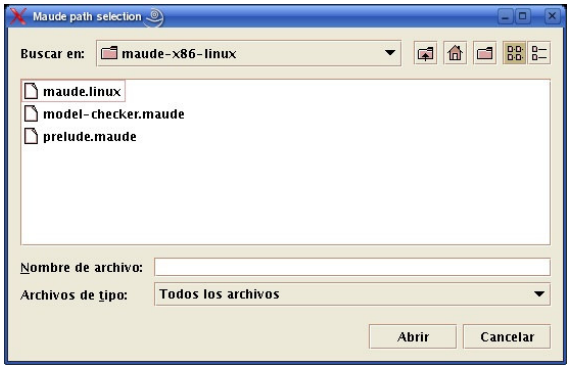
2.2 Iniciando la herramienta

En la primera ejecución de Visual ITP/OCL Tool aparecerá un aviso indicando que las rutas necesarias para la aplicación no se han configurado. Dichas rutas son las de ITP/OCL Tool y el intérprete Maude. El sistema mostrará a continuación una ventana de configuración con una casilla para mostrar cada ruta:



Los botones "*Change*" permiten al usuario realizar la configuración de la ruta correspondiente usando un selector de fichero similar al de la figura siguiente. Para

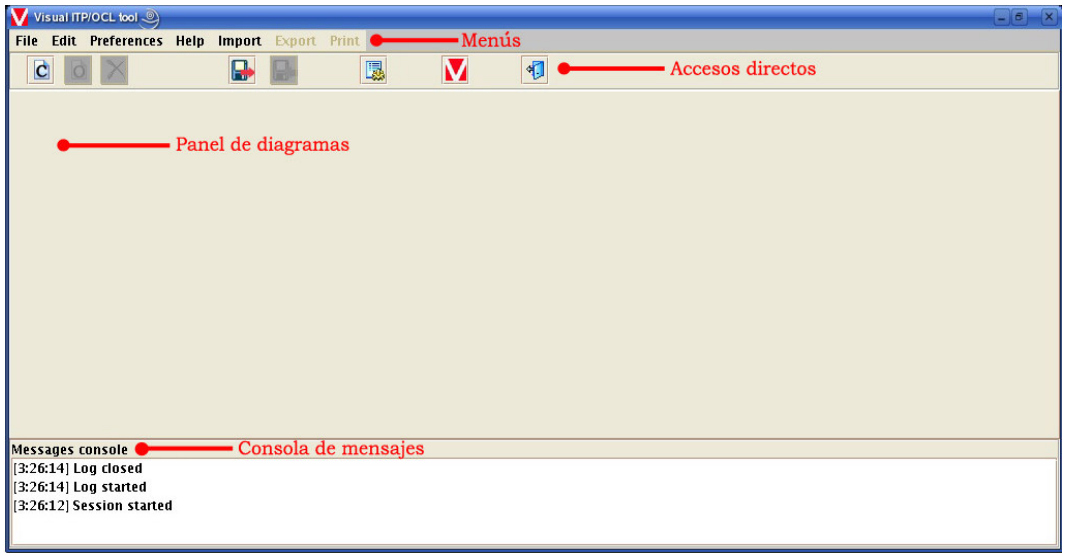
configurar la ruta ITP/OCL Tool será necesario seleccionar el fichero *ocl-tool.maude* de la carpeta donde se encuentre instalado ITP/OCL Tool. Para configurar la ruta de Maude será necesario seleccionar el fichero *maude.linux* de la carpeta donde se encuentre Maude. Una vez seleccionadas las rutas presione "*Accept*" o "*Apply*" y se inicializará Visual ITP/OCL Tool. Si presiona "*Cancel*" no podrá comunicarse con ITP/OCL Tool.



Este proceso de configuración puede repetirse las veces que se desee. Para realizar estas actualizaciones o consultar sus valores puede seleccionar la opción *Configure* en el menú *Preferences*. Las rutas seleccionadas se almacenarán en el fichero *preferencias.txt* existente en el directorio raíz de la aplicación. La primera vez que se ejecuta la herramienta, se crea dicho fichero con las rutas que indique el usuario. Se recomienda no manipular dicho archivo. Cualquier problema existente será notificado al usuario al arrancar la aplicación.

2.3 Aspecto general de la herramienta

Se explican a continuación las distintas partes que ofrece la pantalla de trabajo inicial de Visual ITP/OCL Tool:



Se pueden distinguir 4 bloques de elementos en la pantalla inicial:

1- Menús de la herramienta.
2- Barra de accesos directos.
3- Panel de diagramas.
4- Consola de mensajes.

2.3.1 Menús de la herramienta:

Los menús de la herramienta están asociados a distintos conceptos que serán manejados con frecuencia en el manual, por lo que se detallan a continuación:

- *Diagrama*: cada diagrama independiente creado por el usuario ya sea de clases o de objetos.
- *Conjunto*: grupo de diagramas relacionados entre sí. Un conjunto está compuesto por un único diagrama de clases y cualquier número de diagramas de objetos que sean instancia del diagrama de clases. Los conjuntos no comparten información.

El nombre del conjunto es el de su diagrama de clases al ser este único. El nombre de cada diagrama del conjunto debe ser unívoco, avisando al usuario cuando se asigne un nombre ya existente. Asimismo los conjuntos deben tener nombres distintos. Pueden existir diagramas de igual nombre en distintos conjuntos.

- *Proyecto*: grupo de conjuntos activos en la herramienta. La herramienta solo trabaja con un proyecto a la vez mostrando todos sus conjuntos.

Los distintos menús de la herramienta son:

- *File*: menú encargado de gestionar proyectos en la herramienta:
  - *New*: Crea un proyecto vacío cerrando el proyecto en curso.
  - *Load*: Carga un nuevo proyecto desde la base de datos.
  - *Save*: Almacena un proyecto en la base de datos.
  - *Save as*: Almacena un proyecto nuevo en la base de datos.
  - *Remove*: Cierra el proyecto en curso y lo borra de la base de datos.
  - *Exit*: Salir de la herramienta.

- *Edit*: menú encargado de gestionar los diagramas de los conjuntos:
  - *New*
    - *Class diagram*: Crea un nuevo conjunto en el proyecto con un diagrama de clases vacío.
    - *Object diagram*: Crea un nuevo diagrama de objetos en el conjunto actual.
  - *Restart current diagram*: Elimina todos los elementos dibujados del diagrama actual.
  - *Remove current diagram*: Elimina el diagrama actual del conjunto. Si es el diagrama de clases se elimina el conjunto completo del proyecto.
  - *Copy current class diagram*: Copia el diagrama de clases del conjunto actual en un nuevo conjunto.
  - *Configure current diagram*: Permite cambiar el nombre del conjunto actual, y consecuentemente el del diagrama de clases.

- *Preferences*: menú de configuración de la herramienta.
  - *Configure*: permite cambiar las rutas de ITP/OCL Tool y Maude para realizar las correspondientes actualizaciones.

- *Help*: menú de ayuda para el usuario.
    - *About*: muestra información sobre la herramienta Visual ITP/OCL Tool.

- *Import*: menú de carga de diagramas a partir de distintos formatos.
    - *XML*
        - *Class diagram*: carga un diagrama de clases almacenado en un archivo XML.
        - *Invariants*: carga un conjunto de invariantes asociados a un diagrama de clases almacenados en un archivo XML.
        - *Object diagram*: carga un diagrama de objetos almacenado en un archivo XML.
    - *XMI*
        - *Class diagram*: carga un diagrama de clases almacenado en un archivo XMI.
        - *Object diagram*: carga un diagrama de objetos almacenado en un archivo XMI.

- *Export*: menú de almacenamiento de datos en diversos formatos.
    - *XML*
        - *Class diagram*: almacena el diagrama de clases actual en un archivo XML.
        - *Invariants*: almacena el conjunto de invariantes asociados al diagrama de clases actual en un archivo XML.
        - *Object diagram*: almacena un diagrama de objetos en un archivo XML.
    - *XMI*
        - *Class diagram*: almacena el diagrama de clases actual en un archivo XMI.
        - *Object diagram*: almacena un diagrama de objetos en un archivo XMI.
    - *EPS*
        - *Current diagram*: guarda gráficamente el diagrama actual en un fichero eps con el nombre del diagrama.
        - *Current set*: guarda gráficamente el conjunto actual. Cada diagrama del conjunto se almacena en un fichero eps con su nombre correspondiente.
        - *All diagrams*: guarda los diagramas de todos los conjuntos del proyecto. Cada diagrama se almacena en un fichero eps con su nombre correspondiente.

- *Print*: menú de impresión.
    - *Current diagram*: Imprime el diagrama actual.

NOTA: Algunos menús de la barra superior permanecen inactivos cuando el proyecto está vacío. Esto se debe a que su uso está prohibido en dichas condiciones.

2.3.2 Barra de accesos directos:

La barra principal de la herramienta muestra los siguientes accesos directos:

Cada acceso directo representa una acción de uso frecuente de manera que su ejecución se realice más rápido que usando la barra de menú. Las acciones son:

1 Crear un nuevo diagrama de clases (Edit→New→Class diagram).

2 Crear un nuevo diagrama de objetos (Edit→New→Object diagram).

3 Eliminar el diagrama actual. (Edit→ Remove current diagram)

4 Cargar de la base de datos. (File → Load).

5 Salvar en la base de datos. (File → Save).

6 Configuración de la herramienta. (Preferences→Configure).

7 Información de la herramienta. (Help→About).

8 Salir de la herramienta. (File→Exit).

NOTA: Algunos accesos directos permanecen inactivos cuando el proyecto está vacío. Esto se debe a que su uso está prohibido en esas condiciones.

2.3.3 Panel de diagramas:

El panel central de la ventana principal será el encargado de contener los conjuntos de diagrama de la aplicación. Inicialmente se encuentra vacío al no existir ningún conjunto activo. Para comenzar a trabajar con diagramas consulte el apartado sobre cómo debe crear un diagrama de clases (ver apartado 2.4).

2.3.4 Consola de mensajes:

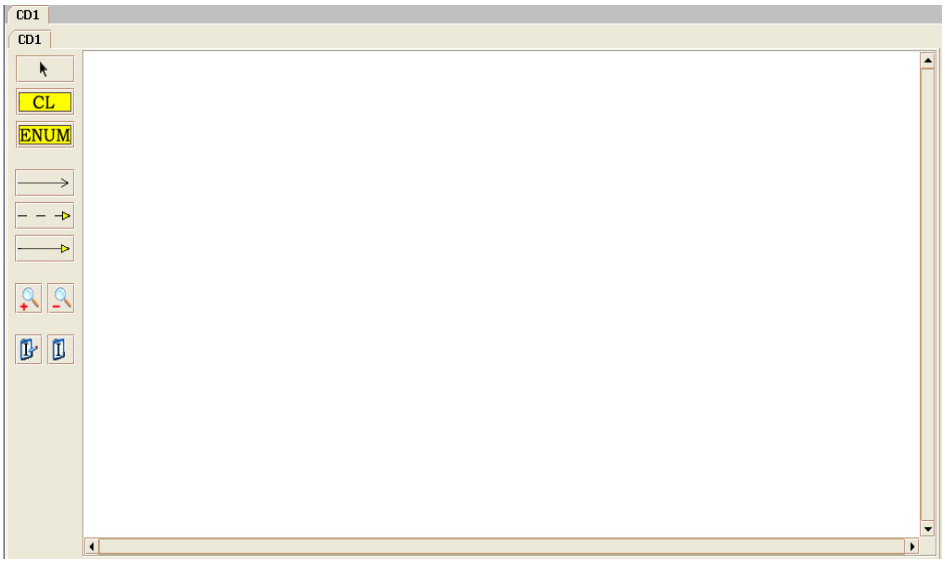
La consola de mensajes muestra al usuario los mensajes provenientes del sistema. Para ello se imprime una línea en la consola con la acción desarrollada, el resultado de la misma, y la hora a la que sucedió el evento. Los mensajes mostrados se corresponden con las respuestas al envío de comandos a ITP/OCL Tool.

2.4 Diagrama de clases

Los diagramas de clases son las piezas fundamentales para construir conjuntos dentro del proyecto. Para crear un diagrama de clases debe elegir Edit→New→ClassDiagram o el correspondiente acceso directo de la barra superior. En ese momento aparecerá un nuevo conjunto con un diagrama de clases vacío y su menú gráfico asociado.

NOTA: Si lo desea puede cargar un diagrama de clases de la base de datos (File→Load o usar su correspondiente acceso directo) o de un fichero XML o XMI (Import→XML→Class Diagram, Import→XMI→Class Diagram). En este caso el diagrama no aparecerá vacío.

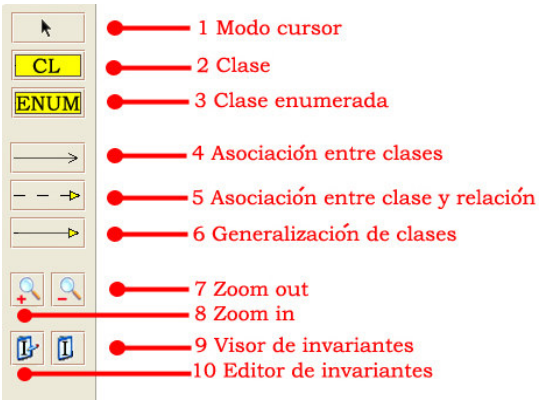
Un diagrama de clases se presenta inicialmente como:



Si es el primer conjunto creado, se activarán los menús y accesos directos que permanecían deshabilitados.

2.4.1 Opciones del diagrama de clases:

Las posibilidades de interacción con un diagrama de clases vienen dadas por su menú asociado (en la parte izquierda de la imagen):



1.- *Botón de modo cursor*: Permite navegar por el diagrama, arrastrar elementos, editarlos y desplegar el menú secundario asociado. Para activar este modo basta con seleccionar el botón de modo cursor y realizar:

- Clic simple en clase o relación: selecciona elemento (aparecerá de otro color).
- Doble clic en clase o relación: edita elemento (ver apartado 2.4.2).
- Botón derecho en clase o relación: despliega menú del elemento.
- Presionar botón izquierdo y arrastrar: mueve una clase o los puntos intermedios de una relación.

2.- *Botón de clase*: Crea una clase en el diagrama. Para ello, presione el botón de clase, y pulse en el punto del diagrama donde desea situar dicha clase. Si lo necesita, puede desplazar las barras de scroll lateral para encontrar el punto deseado. La clase se crea con un nombre por defecto: ClassN, donde N es un entero que comienza en 1 y aumenta

una unidad por cada clase creada. Puede cambiar el nombre editando la clase (ver apartado 2.4.2).

3.- *Botón de clase enumerada*: Crea una clase enumerada en el diagrama. Para ello, presione el botón de clase numerada, y pulse en el punto del diagrama donde desea situar dicha clase. Como las clases enumeradas deben tener como mínimo un atributo, se desplegará la ventana de edición para insertar dicho atributo (ver apartado 2.4.2). Si lo necesita, puede desplazar las barras de scroll lateral para encontrar el punto deseado. La clase se crea con un nombre por defecto: EnumClassN, donde N es un entero que comienza en 1 y aumenta una unidad por cada clase creada. Puede cambiar el nombre editando la clase (ver apartado 2.4.2).

4.- *Botón de asociación entre clases*: Crea una relación de asociación entre dos clases. Para ello, presione el botón de asociación entre clases, haga clic simple sobre la clase origen, y después clic simple sobre la clase destino. Con esto conseguirá una relación recta, dibujada con un único segmento entre las dos clases seleccionadas, y con los valores por defecto. Estos valores serán: AssociationN para el nombre con N un entero comenzando en 1 y aumentando una unidad por cada relación creada; * para las multiplicidades; nombreOrigen@nombreRelacion para el rol origen y nombreDestino @nombreRelacion para el rol destino. Si lo desea puede editar la relación (ver apartado 2.4.2) para cambiar estos valores. Si desea una relación con más de un segmento, debe hacer clic simple sobre la clase origen, hacer clic sobre los puntos del tablero que conformarán los segmentos (se dibujan a medida que hace clic) y pulsar en la clase destino al final.
NOTA: Las clases enumeradas no pueden tomar parte en una relación entre clases.
NOTA: Las relaciones de una clase consigo misma deben tener un mínimo de 3 segmentos.

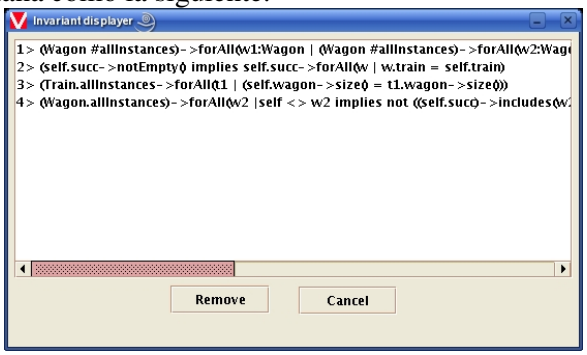
5.- *Botón de asociación entre clase y relación*: Crea una relación de asociación entre una clase de asociación y una relación. Para ello, presione el botón de asociación entre clase y asociación, haga clic simple sobre la clase origen, seleccione los puntos intermedios de la relación (opcional), y haga clic simple en la relación destino. Con esto conseguirá una asociación con los valores por defecto. Estos valores serán: AssociationN para el nombre con N un entero comenzando en 1 y aumentando una unidad por cada relación creada; * para las multiplicidades origen y destino; nombreClaseOrigen@nombreRelacion para el rol origen y nombreRelacionDestino @nombreRelacion para el rol destino. Si lo desea puede editar la asociación (ver apartado 2.4.2) para cambiar estos valores.
NOTA: Las clases numeradas no pueden tomar parte en una relación de asociación.

6.- *Botón de generalización entre clases*: Crea una relación de generalización entre dos clases. Para ello, presione el botón de generalización, haga clic simple sobre la clase origen (clase padre), seleccione los puntos intermedios de la relación (opcional), y haga clic simple en la clase destino (clase hija).
NOTA: Las clases enumeradas no pueden tomar parte en una relación de generalización.

7.- *Botón de zoom out*: Permite reducir la escala del diagrama. Para ello pulse el botón de zoom out repetidas veces hasta obtener el tamaño adecuado.

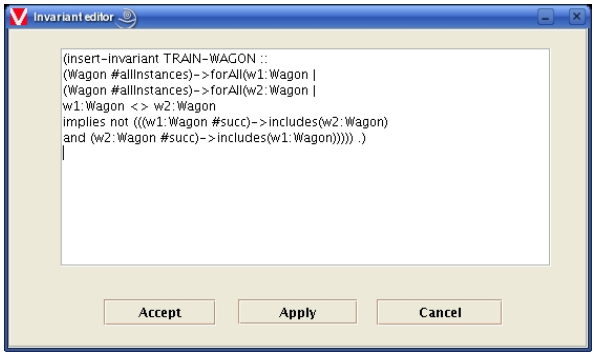
8.- *Botón de zoom in*: Permite agrandar la escala del diagrama. Para ello pulse el botón de zoom in repetidas veces hasta obtener el tamaño adecuado.

9.-*Botón de visor de invariantes*: Muestra los invariantes guardados relacionados con el diagrama de clases actual. Para mostrarlos, pulse el botón de visor de invariantes, y se desplegará una ventana como la siguiente.



Esta ventana le permite consultar los invariantes almacenados y eliminar los que desee. Para eliminar un invariante, selecciónelo de la lista y presione "*Remove*". Una vez terminada la visualización presione "*Cancel*".

10.- *Botón de edición de invariantes*: Permite insertar un invariante relacionado con el diagrama de clases. Existen dos tipos de invariantes: asociados a una clase, y asociados al diagrama. Para insertar un invariante pulse el botón de edición de invariantes, y haga clic simple sobre la clase vinculada al invariante o bien sobre un punto libre del diagrama. Se desplegará un editor como el siguiente:



Tras escribir el invariante puede presionar "*Accept*" o "*Apply*". Si el invariante es sintácticamente correcto, se almacenará en el sistema. Si pulsó "*Accept*" se cerrará el editor, si pulsó "*Apply*" el editor le permitirá insertar otro invariante.

Cuando se inserta un invariante en el sistema, el diagrama de clases se cierra, lo que implica que no podrá modificarlo más. Antes de insertar el primer invariante aparecerá una ventana de confirmación de cierre de diagrama.

Si el invariante es sintácticamente incorrecto, se notificará al usuario y no se almacenará en el sistema.

Puede salir del editor pulsando "*Cancel*".

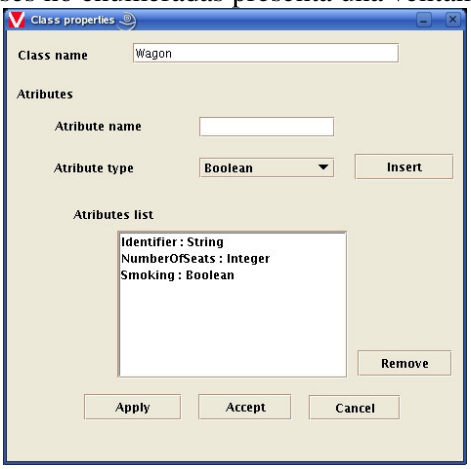
2.4.2 Edición de elementos:

Las clases y relaciones representadas en el diagrama de clases pueden ser editadas en cualquier momento. Para editar un elemento debe hacer doble clic con

botón izquierdo sobre el mismo, o pulsar botón derecho sobre el elemento y seleccionar *Properties* en el menú emergente.

Edición de clases:

La edición de clases no enumeradas presenta una ventana como la siguiente:



El primer campo editable *Class name* muestra el nombre de la clase. Por defecto, la herramienta nombra las clases como ClassN, siendo N un entero que comienza en 1 y aumenta una unidad al crear una clase. Puede cambiar el nombre escribiéndolo en el cuadro de texto.

El resto de la ventana permite editar los atributos de la clase. Para insertar un atributo en la clase, inserte el identificador de atributo en el campo *Atribute name*. Seleccione el tipo deseado del desplegable *Atribute type*, y pulse "*Insert*". Automáticamente el atributo aparecerá en la lista *Atributes lis*t.

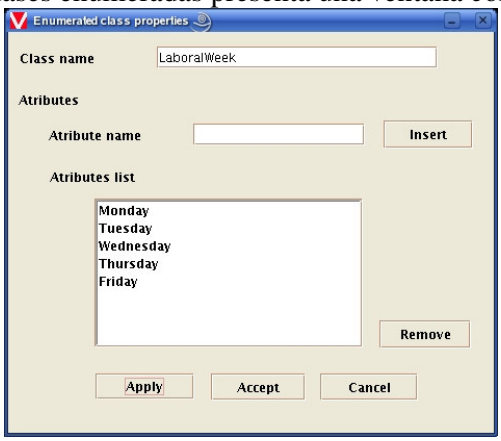
Los tipos por defecto para atributos son enteros (Integer), booleanos (Boolean) y cadenas (String). Si hay clases enumeradas en su diagrama de clases, éstas también aparecerán en el selector de tipos.

Para eliminar un atributo de la lista, selecciónelo con el cursor, y presione *"Remove*". La lista de atributos se actualizará mostrando la eliminación.

Una vez haya concluido la edición de la clase pulse "*Accept*" para guardar el nombre modificado o "*Cancel*" si no desea guardar el cambio de nombre. Si el nombre propuesto es vacío o existe ya en el diagrama, se notificará al usuario para que lo modifique.

Edición de clases enumeradas:

La edición de clases enumeradas presenta una ventana como la siguiente:



El primer campo editable *Class name* muestra el nombre de la clase. Por defecto, la herramienta nombra las clases enumeradas como EnumClassN, siendo N un entero que comienza en 1 y aumenta una unidad al crear una clase. Puede cambiar el nombre escribiéndolo en el cuadro de texto.

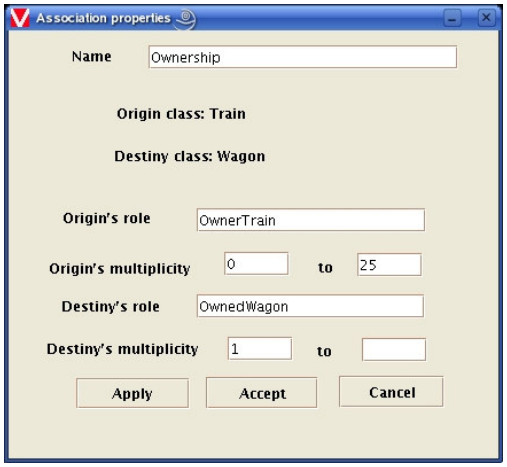
La lista de atributos contendrá los atributos del tipo enumerado. Para insertar un atributo del tipo de la clase, inserte el identificador de atributo en el campo *Atribute name* y pulse "*Insert*". Automáticamente el atributo aparecerá en la lista *Atributes list*.

Para eliminar un atributo de la lista, selecciónelo con el cursor, y presione "*Remove*". La lista de atributos se actualizará mostrando la eliminación.

Una vez haya concluido la edición de la clase pulse "*Accept*" o "*Apply*" para guardar el nombre modificado o "*Cancel*" si no desea guardar el cambio de nombre. Si la lista de atributos o el nombre son vacíos, o el nombre de la clase ya existe en el diagrama, se notificará al usuario para que lo modifique.

Edición de asociaciones:

La edición de asociaciones muestra una ventana con la siguiente:



El primer campo editable *Name* muestra el nombre de la relación. Por defecto los nombres de las relaciones se crean como AssociationN, siendo N un entero que comienza en 1 y aumenta una unidad al crear una relación. Puede cambiar el nombre escribiéndolo en el cuadro de texto.

En segundo lugar se muestran los nombres de la clase origen y destino.

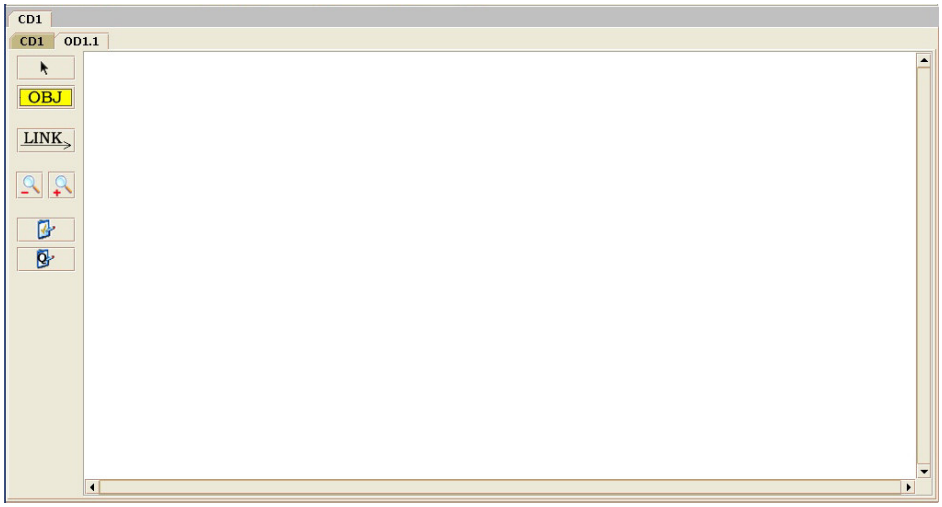
En tercer lugar se muestran los roles y multiplicidades de cada clase dentro de la relación. El campo *Origin's role* muestra el rol de la clase origen. El campo *Destiny's role* muestra el rol de la clase destino. Por defecto los roles de una relación tienen un valor nombreClase@nombreRelacion. Dichos roles se pueden cambiar escribiendo en los cuadros de texto correspondientes.

En cuarto lugar se muestra la multiplicidad de cada clase. El campo *Origin's multiplicity* muestra la multiplicidad de la clase origen. El campo *Destiny's multiplicity* muestra la multiplicidad de la clase destino. Por defecto las multiplicidades tienen el valor * (número ilimitado). Cada multiplicidad se compone de dos campos debido a que las multiplicidades pueden ser simples o compuestas. Las multiplicidades simples se componen de un único valor n con n>0 o n=*. Dicho valor debe escribirse en el campo izquierdo, dejando el derecho vacío. Las multiplicidades compuestas establecen rangos n..m con m>n ó n>0 y m=*. El primer valor debe escribirse en el campo izquierdo y el segundo en el campo derecho.

Para salir y salvar los cambios en nombre, roles y multiplicidades pulse "*Accept*" o "*Apply*". Si se ha cometido algún error se mostrará el mensaje adecuado. Si no desea salvar los cambios pulse "*Cancel*". El nombre de clase, los roles y la multiplicidad son campos obligatorios por lo que se notificará si alguno de estos campos no se ha rellenado.

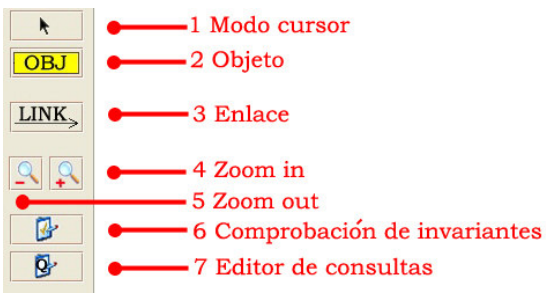
2.5 Diagrama de objetos

Los diagramas de objetos representan situaciones que se pueden derivar de instanciar las clases y relaciones de un diagrama de clases. Por tanto, todo diagrama de objetos necesita partir de un diagrama de clases para crearse. Para crear un diagrama de objetos se debe seleccionar Edit→New→ObjectDiagram o el correspondiente acceso directo de la barra superior. En ese momento aparecerá un nuevo diagrama de objetos vacío en el conjunto actual y su menú gráfico asociado.
NOTA: Si lo desea puede cargar un diagrama de objetos de la base de datos (File→Load o usar su correspondiente acceso directo) o de un fichero XML o XMI (Import→XML→Object Diagram, Import→XMI→Object Diagram). En este caso el diagrama no aparecerá vacío.
NOTA: Al crear un diagrama de objetos se cerrará el diagrama de clases correspondiente. Es aconsejable no crear diagramas de objetos hasta haber terminado el diagrama de clases completamente.
El aspecto inicial de un diagrama de objetos es el siguiente:



2.5.1 Opciones del diagrama de objetos

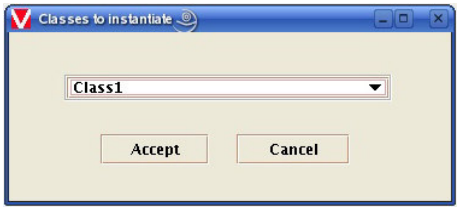
Las posibilidades de interacción con un diagrama de objetos vienen dadas por su menú asociado (en la parte izquierda de la imagen):



1.- *Botón de modo cursor*: Permite navegar por el diagrama, arrastrar elementos, editarlos y desplegar el menú secundario asociado. Para activar este modo basta con seleccionar el botón de modo cursor y realizar:

- Clic simple en objeto o enlace: selecciona elemento (aparecerá de otro color).
- Doble clic en objeto o enlace: edita elemento (ver apartado 2.5.2).
- Botón derecho en objeto o enlace: despliega menú del elemento.
- Presionar botón izquierdo y arrastrar: mueve un objeto o los puntos intermedios de un enlace.

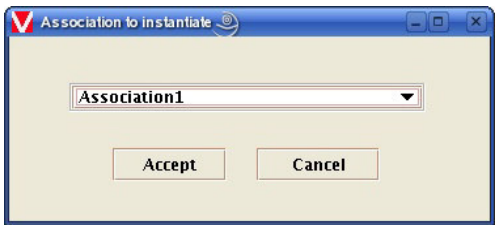
2.- *Botón de objeto*: Crea un objeto en el diagrama. Para ello, presione el botón de objeto, y pulse en el punto del diagrama donde desea situar dicho objeto. Aparecerá un menú desplegable mostrando todas las clases del diagrama de objetos con posibilidad de ser instanciadas. Elija la clase deseada y pulse "*Accept*". Si no desea crear el objeto pulse "*Cancel*".



El objeto creado contendrá los atributos de su clase instanciada y de las clases padre que estén relacionadas con ella a través de una relación de generalización. El nombre del objeto será por defecto ObjectN, siendo N un entero que comienza en 1 y se aumenta en una unidad cada vez que se crea un objeto nuevo. A los atributos se les asignarán valores por defecto: 0 si es entero; true si es booleano; cadena vacía si es string; y el primer elemento de la clase enumerada si es un atributo enumerado. En cualquier momento puede editar el objeto (véase apartado 2.5.2) para cambiar dichos valores.

NOTA: El menú desplegable puede aparecer vacío si el diagrama de clases asociado está vacío o solamente tiene clases no instanciables (clases enumeradas).

3.- *Botón de enlace*: Crea un enlace entre dos objetos o entre un objeto y un enlace. Para ello, presione el botón de enlace, haga clic simple sobre el objeto origen, y por último clic simple sobre el objeto o enlace destino. Aparecerá un menú desplegable mostrando todas las relaciones a instanciar si el enlace es entre dos objetos, pero si el destino es un enlace, el enlace creado se pintará automáticamente al ser este único. Elija la relación de la que desea crear un enlace y pulse "*Accept*". Si no desea crear el enlace pulse "*Cancel*".



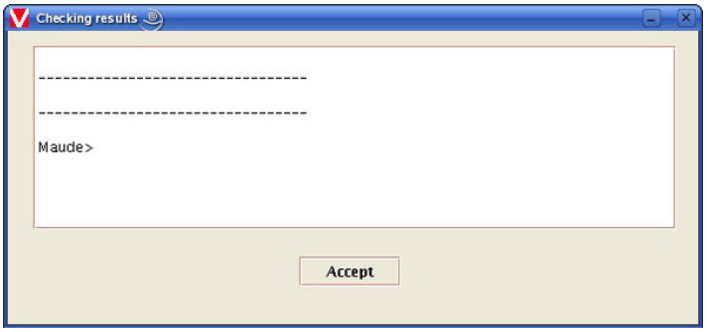
Al igual que para relaciones entre clases este procedimiento crea un enlace con un único segmento. Si desea un enlace con más de un segmento y sin ser una recta, debe hacer clic simple sobre el objeto origen, hacer clic sobre los puntos del tablero que conformarán los segmentos (se dibujan a medida que hace clic) y pulsar en el elemento destino cuando lo desee.

Los enlaces no tienen nombre y sus atributos no son modificables al ser instancias de relaciones. Para consultar los valores de los atributos de los enlaces puede visualizar el enlace (ver apartado 2.5.2),
NOTA: Los enlaces de un objeto en sí mismo deben tener un mínimo de 3 segmentos.

4.- *Botón de zoom in*: Permite agrandar la escala del diagrama. Para ello pulse el botón de zoom in repetidas veces hasta obtener el tamaño adecuado.

5.- *Botón de zoom out*: Permite reducir la escala del diagrama. Para ello pulse el botón de zoom out repetidas veces hasta obtener el tamaño adecuado.

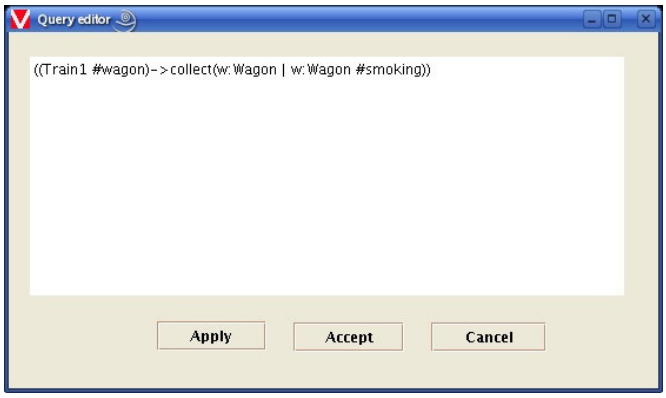
6.- *Botón de comprobación de invariantes*: Permite comprobar los invariantes asociados al diagrama de clases sobre el diagrama de objetos actual. Para ello pulse el botón de comprobación de invariantes. Aparecerá una ventana como la siguiente:



El panel de la ventana mostrará tantos resultados como invariantes había almacenados en el diagrama de clases. Para los invariantes que se cumplan se mostrará OK, y para los que no se cumplan se mostrará FAILED.

Tras consultar los valores pulse "*Accept*" y se cerrará la ventana de comprobación volviendo al diagrama de objetos.

7.- *Botón de edición de consultas*: Permite realizar una consulta sobre el diagrama de objetos actual. Para ello pulse el botón de edición de consultas. Se desplegará un editor como el siguiente:



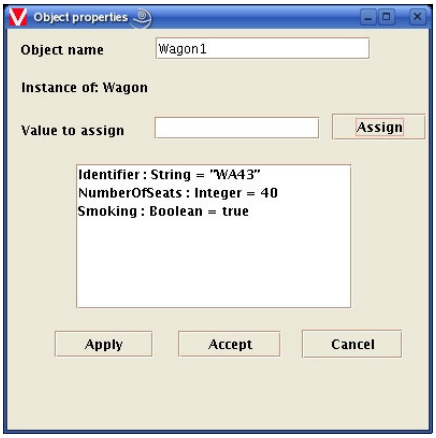
Puede escribir la consulta que desee realizar en el cuadro de texto disponible. Una vez la haya concluido, pulse el botón "*Apply*" para ejecutar la consulta. El resultado se mostrará en el mismo panel de texto. Si no desea formular la consulta pulse "*Accept*" o "*Cancel*".

2.5.2 Edición de elementos

Los objetos y enlaces representados en el diagrama de objetos pueden ser editados en cualquier momento. Para editar un elemento debe hacer doble clic sobre el mismo o pulsar botón derecho sobre el elemento y seleccionar *Properties* en el menú emergente.

Edición de objetos:

La edición de objetos presenta una ventana como la siguiente:



El primer campo editable *Object name* muestra el nombre del objeto. Por defecto, la herramienta nombra los objetos como ObjectN, siendo N un entero que comienza en 1 y aumenta una unidad por cada objeto creado. Puede cambiar el nombre escribiéndolo en el cuadro de texto.

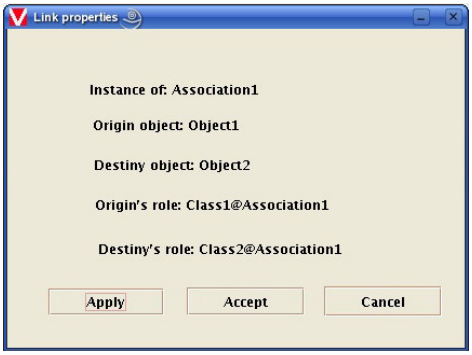
El siguiente campo *Instance of* muestra la clase de la que el objeto es instancia.

La siguiente parte de la ventana permite editar los valores de los atributos del objeto. No se pueden añadir o eliminar atributos pues estos vienen definidos por la clase que se instancia y teniendo en cuenta las relaciones de generalización. Para asignar un valor a un atributo, seleccione el atributo de la lista disponible. Escriba entonces un valor en el campo *Value to assign* y presione "*Assign*". El sistema mostrará ahora el nuevo valor del atributo en la lista de atributos o un mensaje de error si no se pudo realizar la asignación por incompatibilidad de tipos.

Una vez haya concluido la edición del objeto pulse "*Accept*" o "*Apply*" para guardar el nombre modificado. Pulse "*Cancel*" si no desea guardar el cambio de nombre. Si el nombre asignado es vacío, se notificará al usuario puesto que es un campo obligatorio.

Visualización de enlaces:

La visualización de cualquier tipo de enlace muestra una ventana como la siguiente:



Ninguno de los campos mostrados es editable por el usuario. Simplemente se muestra el valor de los atributos a modo informativo.

El primer campo *Instance of* muestra el nombre de la relación del diagrama de clases que se instanció para obtener el enlace actual.

Los siguientes campos *Origin object* y *Destiny object* muestran los nombres de los objetos origen y destino respectivamente.

Los campos *Origin's role* y *Destiny's role* muestran respectivamente los roles del objeto origen y destino del enlace; estos roles coinciden con los de la clase origen y la clase destino en la relación del diagrama de clases.

Una vez terminada la visualización puede salir pulsando "*Accept*", "*Apply*" o "*Cancel*".

3 VISUAL ITP/OCL TOOL

Visual ITP/OCL Tool es una herramienta de desarrollo para la creación de diagramas UML tanto de clases como de objetos y para la comprobación de restricciones sobre diagramas de objetos. Como otros editores de diagramas, Visual ITP/OCL Tool es fácil de usar gracias a su sencillo interfaz gráfico. Bajo este interfaz, se encuentra un potente motor de representación de diagramas y restricciones OCL. Visual ITP/OCL Tool transforma las acciones del usuario sobre el interfaz en comandos que se envían a ITP/OCL Tool. En función del éxito o fracaso de dichos comandos se actualizan los diagramas o se notifica la causa del error al usuario. Esta interacción evita al usuario escribir los comandos en terminal, tarea más compleja que el uso gráfico de la herramienta puesto que requiere un conocimiento de sintaxis muy específica.

Visual ITP/OCL Tool ofrece la posibilidad de trabajar con invariantes y consultas. Además, presenta herencia de atributos y relaciones entre clases, posibilidad que no presentan otras herramientas comerciales. La comparativa de Visual ITP/OCL frente a otras herramientas se puede consultar en el apartado 3.3.

Además de las posibilidades que ofrece su motor de razonamiento, Visual ITP/OCL Tool abarca utilidades de editores profesionales como: carga y almacenamiento en ficheros XML y XMI compatibles con otras aplicaciones, posibilidad de cargar y almacenar en base de datos, posibilidad de guardar en archivos gráficos Eps, posibilidad de interactuar dinámicamente con los diagramas, aplicar zoom a los diagramas o imprimir diagramas.

3.1 Arquitectura del sistema

La arquitectura interna de Visual ITP/OCL Tool sigue unos patrones de diseño con el objetivo de hacer más comprensible y sencillo el código.

Existe una clara separación entre parte lógica y parte gráfica. Para cualquier elemento que necesite una representación, por ejemplo un diagrama de clases, se trabaja a dos niveles. El nivel lógico implementa las acciones propiamente dichas sobre las estructuras de datos: añadir clases, añadir relaciones, insertar un atributo en una clase… La parte gráfica muestra la situación de las estructuras de datos, pero en ningún momento las modifica. Si la parte gráfica necesita en algún momento modificar las estructuras de datos (por ejemplo debido a eventos de ratón) delega la acción a la parte lógica, la cual actualizará las estructuras convenientemente. Esta separación permite poder variar un nivel sin afectar al siguiente lo cual es muy útil a la hora de realizar modificaciones o mejores en una parte concreta.

En segundo lugar se ha considerado una estructura jerárquica basada en fachadas. Una fachada no es más que un punto centralizado de acceso a una parte del sistema que incluye dentro de sí distintos componentes y funcionalidades a las que puede acceder el usuario, pero no de cualquier forma, sino exclusivamente a través de la fachada. La fachada maneja por tanto los distintos componentes y realiza sobre ellos las acciones que desea el usuario. Estos componentes pueden ser a su vez fachadas internas o elementos máximos de una jerarquía. En Visual ITP/OCL Tool la fachada principal es la ventana principal que contiene los menús de la aplicación y los conjuntos de diagramas. Esta fachada permite acceder a tres tipos de elementos: menús, diagramas de clases y diagramas de objetos. Las peticiones de usuario se direccionan al que sea conveniente en cada momento. A su vez los

diagramas son una estructura compuesta por elementos de tipo clase u objeto y elementos de tipo relación o enlace, todos los cuales pueden tener subcomponentes como atributos o textos flotantes. Las acciones que se envían a los diagramas se direccionan a las estructuras de sus elementos y a los propios elementos si es necesario.

Con estas ideas de jerarquización y separación se ha desarrollado el código de la aplicación intentando facilitar el trabajo de ampliación o modificación que se puede realizar sobre él en el futuro. A este fin se han distinguido varios paquetes dentro de la aplicación. Los paquetes recogen clases con un fin común:

- Paquete Dibujables.
- Paquete Interfaz.
- Paquete Eps.
- Paquete ModBD.
- Paquete Conexión.

A continuación desglosamos las clases de cada paquete para ofrecer una visión del objetivo del mismo. Se presentarán los diagramas de cada paquete para obtener una idea global de la aplicación completa.

3.1.1 Paquete Dibujables:

El paquete Dibujables recoge los elementos básicos que se encuentran en cada diagrama: clases, objetos, relaciones, enlaces y elementos auxiliares necesarios para clases y objetos como son atributos y textos flotantes. Los cuatro primeros elementos tienen una función muy importante que cumplir como es la de dibujarse en el diagrama. Cuando un diagrama se dibuja sobre el tapiz se llama a la función de pintado de cada elemento del diagrama.

- *ClassGUI*: Representa a las clases de un diagrama de clases. Entre sus atributos más importantes se encuentra el nombre de la clase, las coordenadas de posición en el tapiz, un valor que indica si es enumerada o no, y un vector de atributos para representar los atributos de clase. Cuenta también con atributos que facilitan el pintado, así como la mayoría de los métodos. Los dos métodos principales son los de pintado y de envío a Maude.

- *ObjectGUI*: Representa a los objetos de un diagrama de objetos. Entre sus atributos más importantes se encuentra el nombre del objeto, el nombre de la clase instanciada, las coordenadas de posición en el tapiz y un vector con atributos. A diferencia de las clases, en los objetos los atributos del vector tienen siempre un valor. En las clases no es necesario. Los demás atributos ayudan al pintado, así como la mayoría de métodos. Como todos los elementos dibujables, los métodos más importantes son el de pintado y envío a Maude.

- *RelationGUI*: Representa a las relaciones entre clases de un diagrama de clases. Entre sus atributos más importantes destacan el nombre de la relación, los roles y multiplicidades, las clases origen y destino o la relación destino y el vector de segmentos. El tratamiento y pintado de las relaciones es más complicado que el de clases, por lo que necesita más métodos auxiliares. Tanto para el pintado como

para el envío a Maude se debe diferenciar si trabajamos con una asociación clase-clase, asociación clase-relación o generalización. Esto se lleva a cabo realizando comprobaciones en atributos especiales.

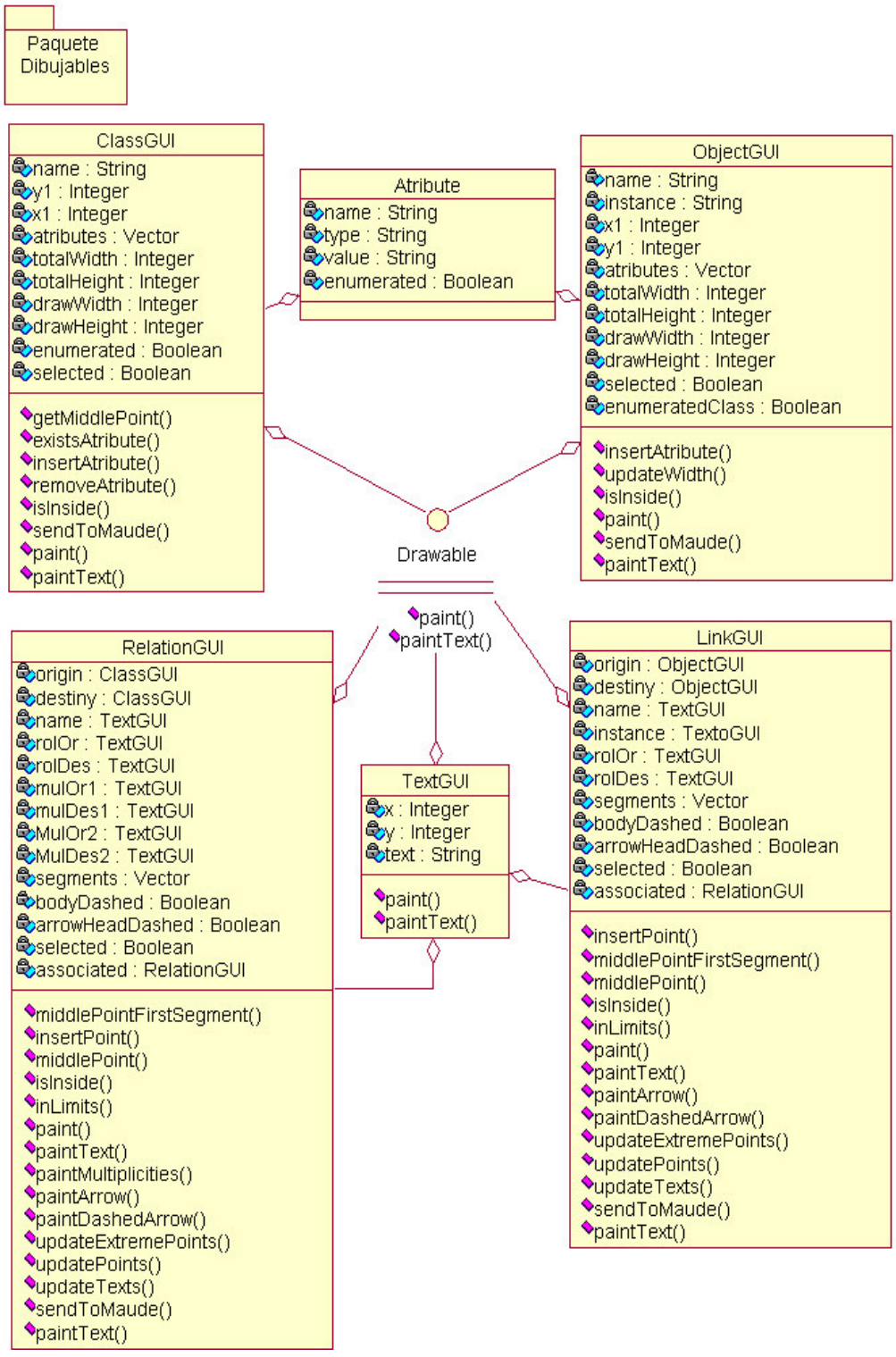
- *LinkGUI*: Representa a los enlaces entre objetos de un diagrama de objetos. Sus atributos principales son nombre, roles, objetos origen y destino o el enlace destino y el vector de segmentos. Al igual que las relaciones, el pintado y manejo de los enlaces es complicado por lo que necesitan métodos auxiliares. El pintado y el envío a Maude son los métodos más importantes, teniendo en cuenta si el destino se trata de un objeto o de otro enlace.

- *Atribute*: Representa a los atributos de una clase u objeto. Vienen definidos por tres atributos básicos: nombre, tipo y valor, y un atributo que indica si proceden de clase enumerada. Cuando trabajamos con clases, no es necesario que los atributos tengan valor, pero para los objetos es obligatorio.

- *TextGUI*: Representa texto dibujable utilizado en relaciones y enlaces para almacenar el nombre, roles y multiplicidades. Son cadenas de texto con coordenadas para actualizarse, que responden a los movimientos de las relaciones. Tener esta clase es más sencillo que tener cadenas de texto corriente en las relaciones o enlaces, lo que provocaría problemas a la hora de actualizar posiciones.

- *Drawable*: Es el interfaz que debe implementar todo elemento GUI que se dibuje a si mismo. Debe implementar el dibujo del elemento y del texto asociado si lo requiere (para relaciones y enlaces).

El diagrama de clases completo del paquete es el siguiente:



3.1.2 Paquete interfaz:

El paquete interfaz recoge todos los elementos gráficos con los que se puede interactuar en la herramienta: diagramas, ventanas, menús… Al ser una herramienta puramente visual, este paquete contiene la mayor parte de los ficheros.

- *RightMenuClass*, *RightMenuRelation*, *RightMenuObject*, *RightMenuLink*, *BasicMenu*: Son menús desplegables emergentes asociados a los elementos de los diagramas: clases, relaciones, objetos y enlaces. Aparecen al presionar el botón derecho sobre el elemento deseado. Las opciones mostradas son cortar, copiar, editar y borrar. El menú básico es desplegado sobre una zona vacía para pegar un elemento previamente cortado o pegado.

- *Theme*: Presenta una configuración visual propia de Visual ITP/OCL Tool basada en juegos de colores y fuentes.

- *Editor*, *DisplayerWindow*: *Editor* despliega un editor de invariantes. Los invariantes se editan y almacenan en el diagrama de clases asociado si son correctos. *DisplayerWindow* muestra todos los invariantes almacenados en el diagrama.

- *Main*: Archivo principal para lanzar Visual ITP/OCL Tool.

- *QueryEditor*: Despliega un editor de consultas sobre un diagrama de objetos. Permite escribirlas, ejecutarlas y observar los resultados.

- *CheckingWindow*: Ventana que muestra los resultados obtenidos de comprobar sobre un diagrama de objetos los invariantes almacenados en el diagrama de clases asociado.

- *ClassMenu*, *EnumClassMenu*, *RelationMenu*, *ObjectMenu*, *LinkMenu*: Despliegan ventanas de edición referidas al elemento deseado: clase, asociación, objeto o enlace. Dichas ventanas permiten cambiar el valor de ciertos atributos como nombre, rol, multiplicidad… Para clases se pueden añadir atributos de clase, y para objetos se puede dar valores a los atributos.

- *ClassSelectionMenu*, *RelationSelectionMenu*: Muestran menús desplegables con las posibles clases o relaciones que se pueden instanciar en un diagrama de objetos cuando se desea crear un elemento.

- *PreferencesMenu*: Muestra una ventana de configuración para Visual ITP/OCL Tool donde se pueden seleccionar las ruta de ITP/OCL Tool y Maude.

- *PropertiesMenu*: Muestra una ventana de configuración que permite cambiar el nombre del diagrama en uso.

- *Principal*, *PrincipalGUI*: Principal contiene todas las acciones posibles a realizar sobre la ventana principal de trabajo, esto es, acciones sobre los menús y accesos directos y manejo de las pestañas de los diagramas. Sus principales funciones son carga y almacenamiento de conjuntos en base de datos, crear y trabajar con diagramas, cargar y almacenar diagramas en formato XML, XMI y EPS, e imprimir diagramas. *PrincipalGUI* contiene todos los elementos de interfaz gráfica necesarios para llevar a cabo las funciones de *Principal*. Cabe destacar el almacenamiento de los diagramas individuales dentro del marco principal. Para ello, se utilizan dos niveles gráficos de pestañas. El nivel exterior representa el conjunto del diagrama. El nivel interior contiene tantas pestañas como diagramas tiene el conjunto. Las pestañas son elementos TabbedPane con diagramas como contenido. Como los diagramas son a su vez componentes gráficas es sencillo integrarlas en el marco principal.

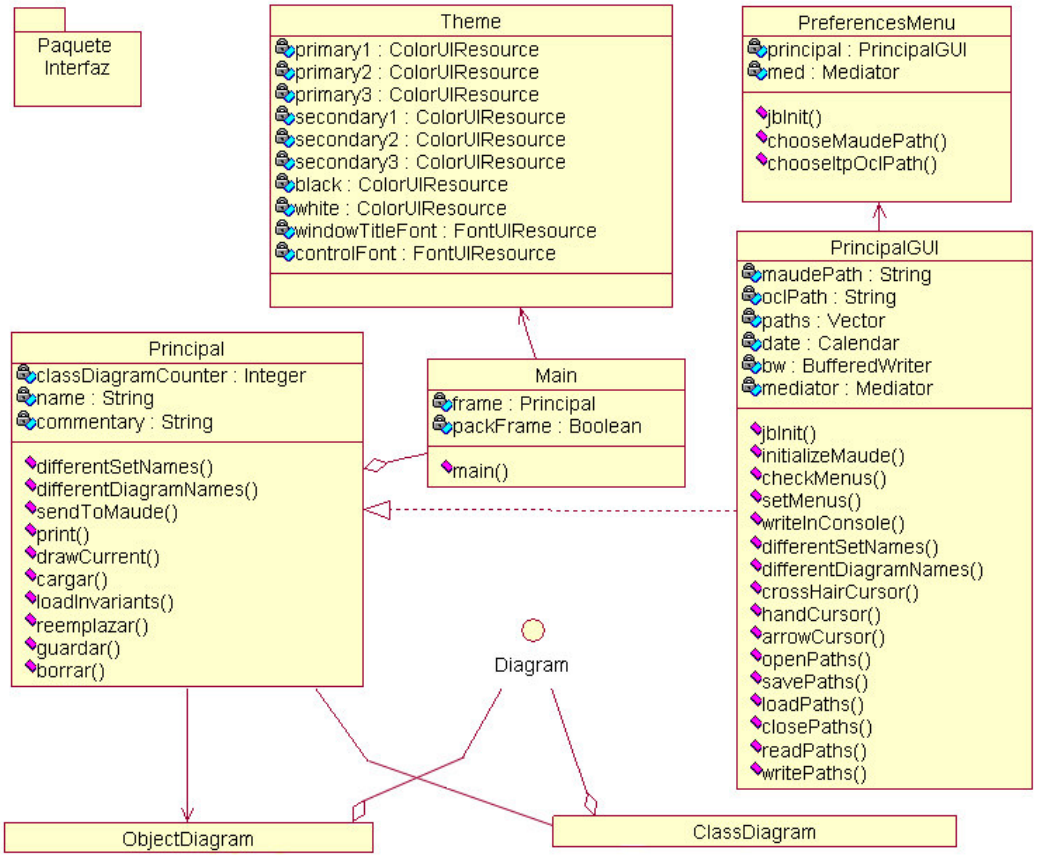
- *ClassDiagram*, *ClassDiagramGUI*, *ObjectDiagram*, *ObjectDiagramGUI*: Representan cada tipo de diagrama (clases y objetos) junto con sus representaciones gráficas (GUI). Los diagramas permiten crear elementos del tipo correspondiente y relaciones entre ellos. Dichos elementos se pueden editar posteriormente por el usuario. Para los diagramas de clase existen además invariantes, del mismo modo que para los de objetos existen consultas. La estructura interna de los diagramas es la misma en los dos casos. Las estructuras de datos son vectores, teniendo uno para elementos y otro para relaciones. Se accede a los vectores usando un índice que se corresponde con el elemento que selecciona el usuario. Para poder calcularlo, se han implementado los métodos de evento de ratón correspondientes a clickear, presionar, arrastrar y soltar. Estos eventos se han implementado sobre una clase de panel que extiende a la clase Panel de Java. Esta extensión era obligatoria para poder implementar un método de pintado acorde a los objetivos de la herramienta. Dicho método recorre los vectores de elementos y relaciones y llama al método de pintado de cada uno, como se explicó en los elementos del paquete dibujables. Los diagramas implementan además operaciones de inserción, eliminación, carga y almacenamiento en distintos formatos, zoom, envío a Maude… por lo que se requieres bastantes variables auxiliares como se puede comprobar en los ficheros correspondientes.

- *Diagram*: Indica las operaciones obligatorias que deben implementar los diagramas de clases y objetos.

- *Load*, *Delete*, *LoadMenu*: Representan archivos necesarios para trabajar a nivel gráfico con la base de datos. Respectivamente sirven para cargar diagramas, borrar diagramas y cargar conjuntos.

El diagrama de clases completo se desglosa en 3 partes principales: principal, diagramas de clases y diagramas de objetos.

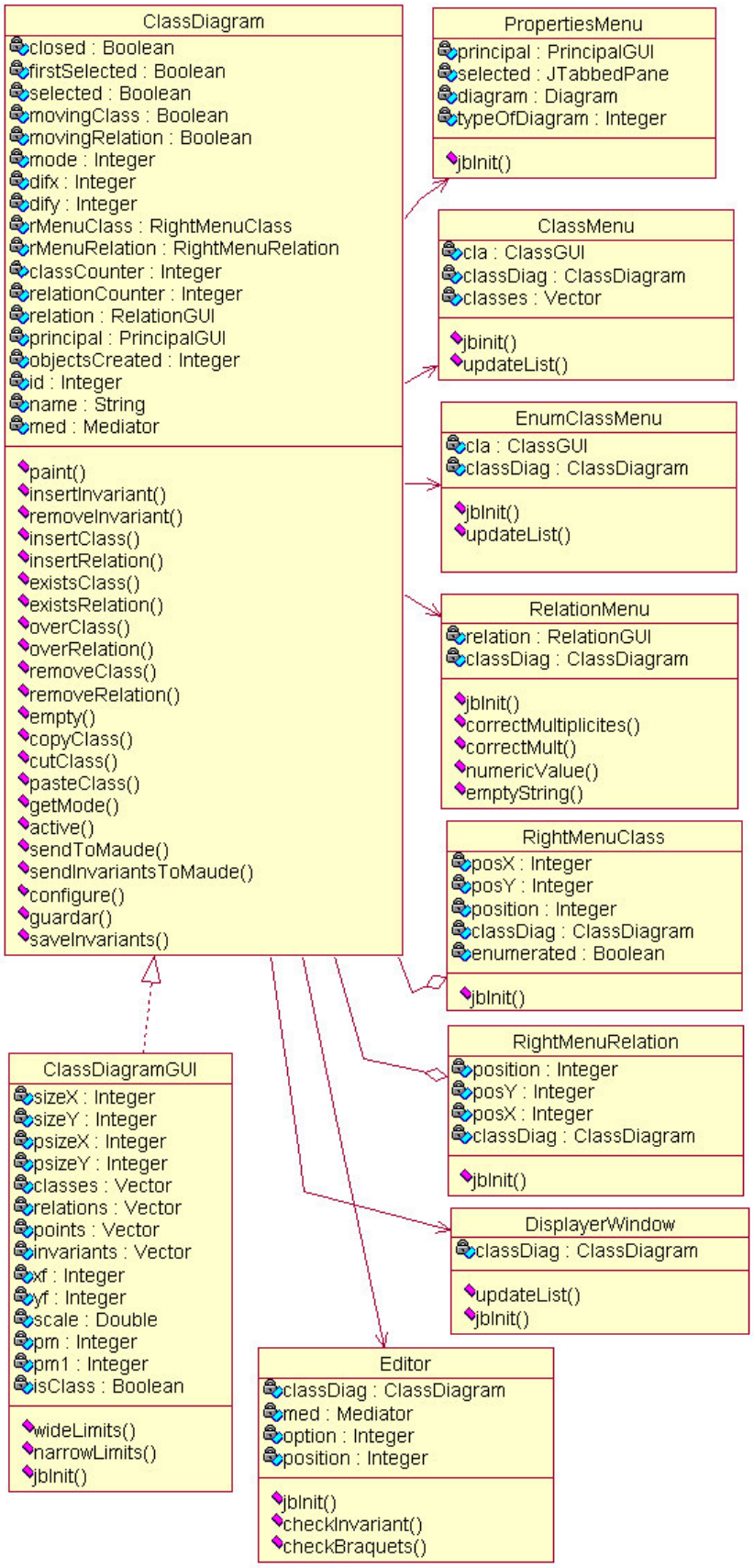
Vemos como Principal es la fachada del sistema, incluyendo dentro de sí diagramas de clases y de objetos a la vez que funciones generales sobre ellos.

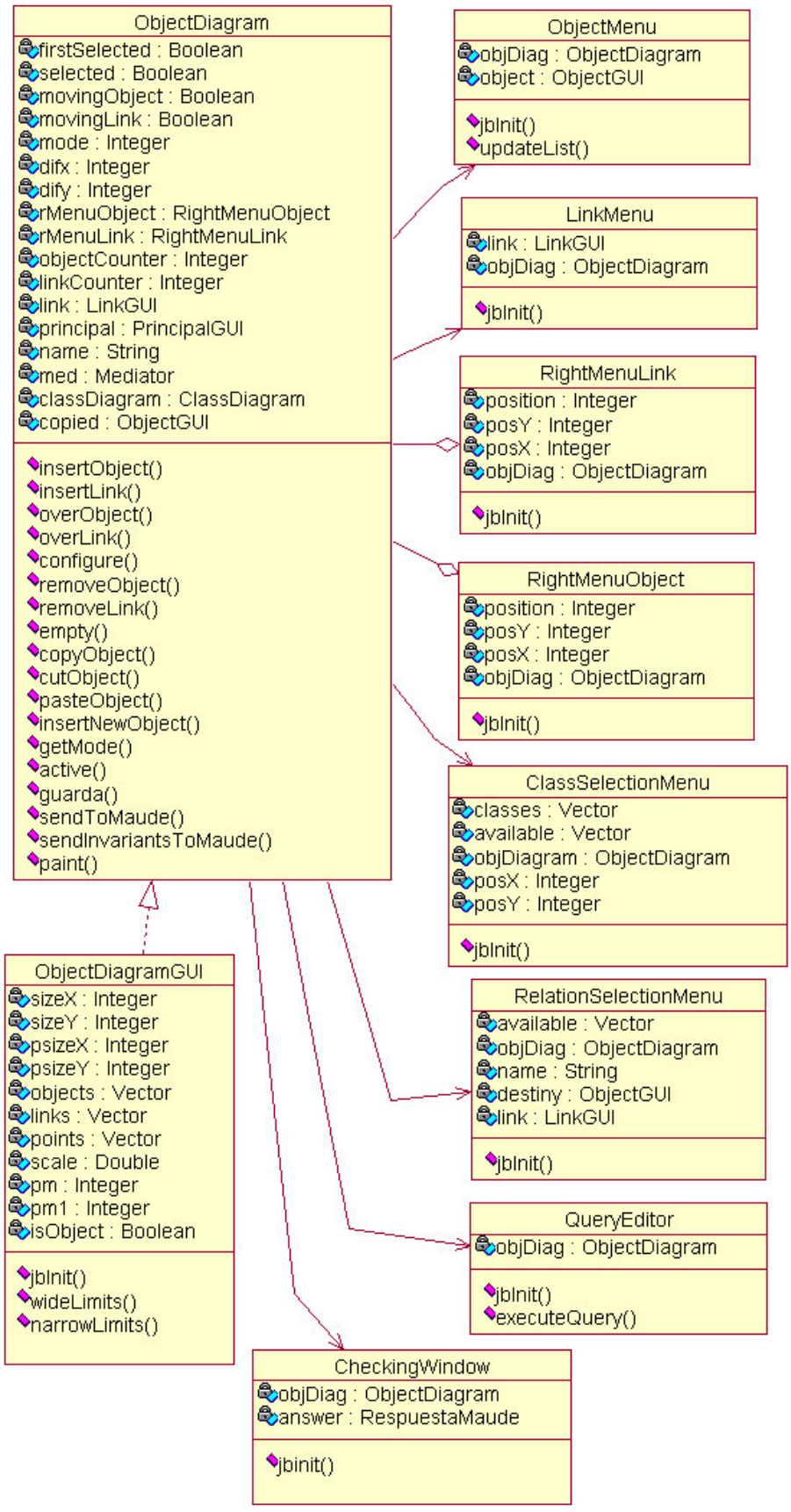


Los diagramas de clases y de objetos a su vez se descomponen en elementos (clases, objetos) y relaciones (asociaciones, enlaces) de manera jerarquizada. Estos elementos pueden hacer uso de elementos de un nivel jerárquico inferior como atributos o textos flotantes.

Cada diagrama tiene su correspondiente interfaz gráfico que delega las operaciones a la parte lógica. Esta parte lógica puede delegar en funciones sobre los elementos que posee para no sobrecargarse de funcionalidad. Los diagramas necesitan algunos elementos visuales secundarios, necesarios para la ejecución de la herramienta, como menús derechos, ventanas de edición o editores de invariantes (diagrama de clases) o consultas (diagrama de objetos).

Las páginas siguientes muestran respectivamente el diagrama de clases referido a los diagramas de clases de la aplicación y diagramas de objetos de la aplicación.


ClassDiagram
closed : Boolean
firstSelected : Boolean
selected : Boolean
movingClass : Boolean
movingRelation : Boolean
mode : Integer
difx : Integer
dify : Integer
rMenuClass : RightMenuClass
rMenuRelation : RightMenuRelation
classCounter : Integer
relationCounter : Integer
relation : RelationGUI
principal : PrincipalGUI
objectsCreated : Integer
id : Integer
name : String
med : Mediator
paint()
insertInvariant()
removeInvariant()
insertClass()
insertRelation()
existsClass()
existsRelation()
overClass()
overRelation()
removeClass()
removeRelation()
empty()
copyClass()
cutClass()
pasteClass()
getMode()
active()
sendToMaude()
sendInvariantsToMaude()
configure()
guardar()
saveInvariants()
PropertiesMenu
principal : PrincipalGUI
selected : JTabbedPane
diagram : Diagram
typeOfDiagram : Integer
jbInit()
ClassMenu
cla : ClassGUI
classDiag : ClassDiagram
classes : Vector
jbinit()
updateList()
EnumClassMenu
cla : ClassGUI
classDiag : ClassDiagram
jbInit()
updateList()
RelationMenu
relation : RelationGUI
classDiag : ClassDiagram
jbInit()
correctMultiplicites()
correctMult()
numericValue()
emptyString()
RightMenuClass
posX : Integer
posY : Integer
position : Integer
classDiag : ClassDiagram
enumerated : Boolean
jbInit()
RightMenuRelation
position : Integer
posY : Integer
posX : Integer
classDiag : ClassDiagram
jbInit()
DisplayerWindow
classDiag : ClassDiagram
updateList()
jbInit()
ClassDiagramGUI
sizeX : Integer
sizeY : Integer
psizeX : Integer
psizeY : Integer
classes : Vector
relations : Vector
points : Vector
invariants : Vector
xf : Integer
yf : Integer
scale : Double
pm : Integer
pm1 : Integer
isClass : Boolean
wideLimits()
narrowLimits()
jbInit()
Editor
classDiag : ClassDiagram
med : Mediator
option : Integer
position : Integer
jbInit()
checkInvariant()
checkBraquets()


ObjectDiagram
firstSelected : Boolean
selected : Boolean
movingObject : Boolean
movingLink : Boolean
mode : Integer
difx : Integer
dify : Integer
rMenuObject : RightMenuObject
rMenuLink : RightMenuLink
objectCounter : Integer
linkCounter : Integer
link : LinkGUI
principal : PrincipalGUI
name : String
med : Mediator
classDiagram : ClassDiagram
copied : ObjectGUI
insertObject()
insertLink()
overObject()
overLink()
configure()
removeObject()
removeLink()
empty()
copyObject()
cutObject()
pasteObject()
insertNewObject()
getMode()
active()
guarda()
sendToMaude()
sendInvariantsToMaude()
paint()
ObjectMenu
objDiag : ObjectDiagram
object : ObjectGUI
jbInit()
updateList()
LinkMenu
link : LinkGUI
objDiag : ObjectDiagram
jbInit()
RightMenuLink
position : Integer
posY : Integer
posX : Integer
objDiag : ObjectDiagram
jbInit()
RightMenuObject
position : Integer
posY : Integer
posX : Integer
objDiag : ObjectDiagram
jbInit()
ClassSelectionMenu
classes : Vector
available : Vector
objDiagram : ObjectDiagram
posX : Integer
posY : Integer
jbInit()
ObjectDiagramGUI
sizeX : Integer
sizeY : Integer
psizeX : Integer
psizeY : Integer
objects : Vector
links : Vector
points : Vector
scale : Double
pm : Integer
pm1 : Integer
isObject : Boolean
jbInit()
wideLimits()
narrowLimits()
RelationSelectionMenu
available : Vector
objDiag : ObjectDiagram
name : String
destiny : ObjectGUI
link : LinkGUI
jbInit()
QueryEditor
objDiag : ObjectDiagram
jbInit()
executeQuery()
CheckingWindow
objDiag : ObjectDiagram
answer : RespuestaMaude
jbinit()

3.1.3 Paquete Eps:

El paquete Eps tiene el cometido específico de guardar cualquier tipo de diagrama en un archivo Eps (Encapsulated Post Script). Para ello se ha utilizado una librería de libre distribución denominada Jibble (www.jibble.org).

La mayor ventaja que proporciona esta librería es que permite tratar los documentos eps como elementos gráficos de Java, es decir elementos de tipo Graphics2D. Como todos los elementos gráficos de la herramienta (clases, objetos, relaciones, enlaces) se dibujan sobre el diagrama usando métodos Graphics2D la conversión de diagrama a formato eps es inmediata al compartir el algoritmo de pintado.

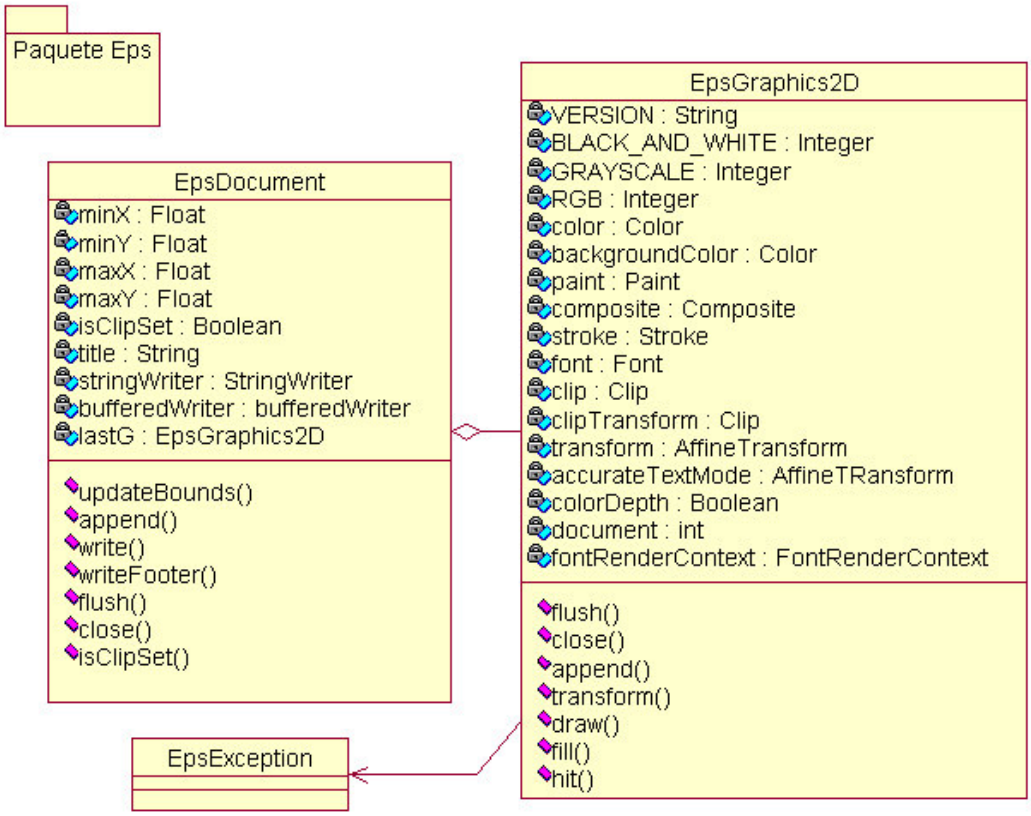
La librería Jibble fue adaptada en ciertos puntos para ser más eficaz en la herramienta, por lo que sus archivos pasaron de estar en la librería a ser archivos de la herramienta.

- *EpsDocument*: Representa el archivo del diagrama. Implementa las funciones usuales de un archivo: abrir, cerrar, vaciar, concatenar, limpiar…

- *EpsGraphics2D*: Es el archivo básico para guardar en formato eps. Representa los dibujos que pueden ser almacenados en un documento, por lo que sobrescribe todos los métodos de la clase Graphics2D. Este elemento se pasa como parámetro a los métodos de pintado de la herramienta para transcribir los diagramas sobre él.

- *EpsException*: Excepción propia de conversión a eps.

El diagrama de clases se presenta a continuación:



3.1.4 Paquete conexión:

El paquete conexión realiza la comunicación entre Visual ITP/OCL Tool e ITP/OCL Tool a través del intérprete Maude. Dicha comunicación se realiza por medio de comandos que manda la herramienta a un conector específico. El conector ejecuta este comando en Maude, y devuelve a la herramienta una respuesta para ser procesada. Si la respuesta es positiva, se actualizan los diagramas, y si es negativa se avisa al usuario del error.

El paso de comandos se hace ejecutando cadenas de texto a través de terminal utilizando el comando *bash*. Este es el método más sencillo de ejecución en los sistemas Linux aunque no el más eficiente. Tanto los diagramas como los elementos contenidos en ellos son capaces de generar los comandos de su creación utilizando la sintaxis de comandos ITP/OCL Tool junto con datos que se consideren oportunos. Para mandar un diagrama completo a Maude, basta con mandar el comando de creación del diagrama y a continuación recorrer las estructuras de datos de elementos mandando el comando de cada elemento concreto a Maude. En cualquier momento que un elemento sea modificado se manda el comando de dicho elemento para actualizar ITP/OCL Tool.

Los comandos enviados a Maude se registran en un archivo denominado *log.txt* situado en el directorio raíz de modo que el usuario pueda consultarlo si lo desea.

- *Maude*: Interactúa con el intérprete Maude realizando las mismas acciones que un usuario puede ejecutar interactuando con Maude a través de terminal. Entre ellas destacan arrancar ITP/OCL Tool y ejecutar sus comandos.

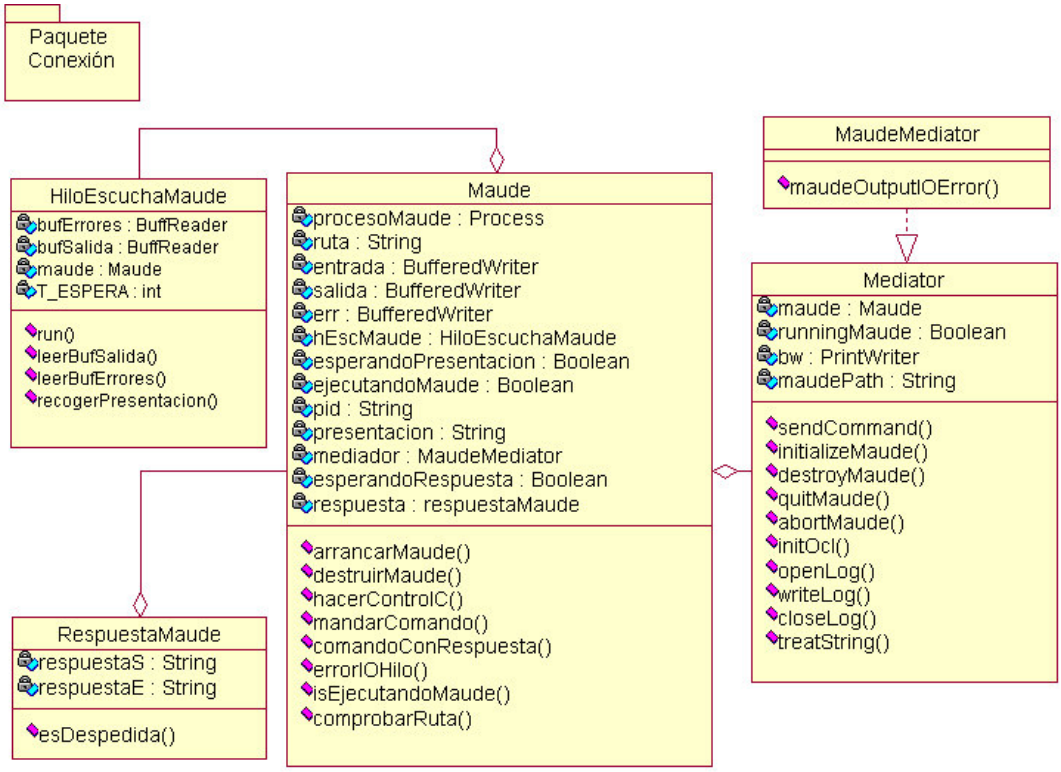
- *HiloEscuchaMaude*: Utilizado por Maude para conocer el estado del mismo y capaz de actuar cuando sucede un evento.

- *MaudeMediator*: Clase a extender con las acciones imprescindibles que debe implementar el mediador que enlace Visual ITP/OCL Tool con Maude.

- *Mediator*: Representa el elemento real que hará la mediación entre Visual ITP/OCL Tool y Maude. Este elemento debe aparecer obligatoriamente en todas las clases de la herramienta que necesiten interactuar con Maude. Está encargado de realizar todas las acciones: arrancar Maude, arrancar ITP/OCL Tool, ejecutar comandos, devolver respuestas… Para ello usa los elementos citados previamente.

- *RespuestaMaude*: Cadena que se obtiene como respuesta al envío de un comando a ITP/OCL Tool utilizando Maude. Estas cadenas deben verificarse para ver si indican éxito (Ok), error (Failed) o error de Maude (cadena vacía). En función de la respuesta la herramienta realizará una función determinada.

El diagrama de clases del paquete es el siguiente:



3.1.5 Paquete modBD:

El paquete modBD se encarga de ejecutar las acciones sobre la base de datos de la aplicación. Esta base de datos está diseñada en MySQL y es capaz de almacenar todos los conjuntos y diagramas del usuario. Del mismo modo se pueden cargar en la aplicación o ser borrados a petición del usuario.

La base datos está pensada para ser un almacén permanente de diagramas de modo que el usuario pueda recurrir a ella siempre que lo desee. Para compartir estos diagramas basta con cargarlos en la aplicación y convertirlos a formato XML, XMI o Eps. Del mismo modo un usuario puede adquirir diagramas de otro cargándolos en la aplicación y guardándolos posteriormente en la base de datos.

El uso de la base de datos no es obligatorio a la hora de utilizar la herramienta, pero sí recomendable para tener una copia de seguridad en caso de fallo en los archivos intercambiados. Para crearla se debe ejecutar la secuencia de comandos SQL que se incluye en la distribución.

Los archivos del paquete se corresponden con la estructura de los diagramas:

- *BaseDatos*, *IBaseDatos*: Encargados de realizar la conexión inicial con la base de datos.

- *BaseDatosFiles*, *IBaseDatosFiles*: Encargados de almacenar, cargar y borrar conjuntos de diagramas.

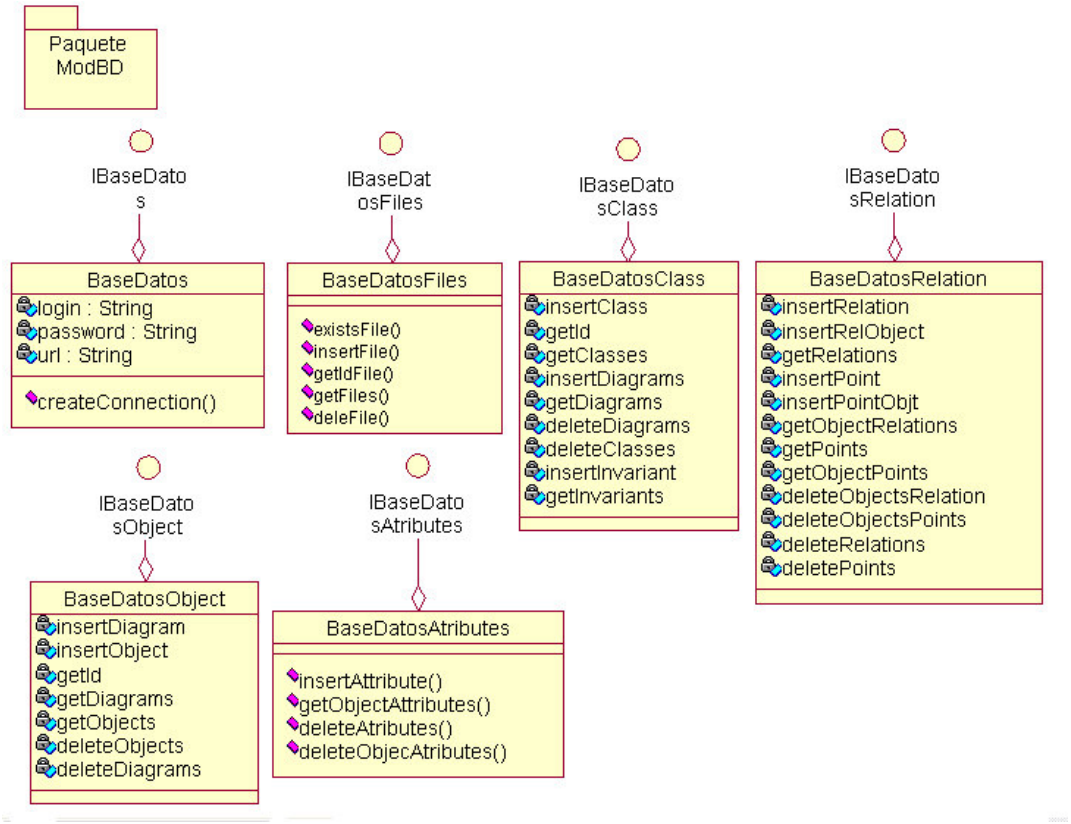
- *BaseDatosClass*, *IBaseDatosClass*: Encargados de almacenar, cargar y borrar diagramas de clases e invariantes.

- *BaseDatosObject*, *IBaseDatosObject*; Encargados de almacenar, cargar y borrar diagramas de objetos.

- *BaseDatosRelations*, *IBaseDatosRelations*: Encargados de almacenar, cargar y borrar relaciones y enlaces en los distintos tipos de diagramas.

- *BaseDatosAtributes*, *IBaseDatosAtributes*: Encargados de almacenar, cargar y borrar atributos en los distintos tipos de diagramas.

  El diagrama de clases del paquete es el siguiente:



3.2 Historial de reuniones

Reunión 1, 10-10-05:

Primera reunión formal para tratar las bases de la aplicación. Se comenzará realizando un editor simple de diagramas de clases que permita:

- Crear clases y trabajar con su nombre y atributos.
- Crear asociaciones y trabajar con sus clases origen y destino.

Los dibujos necesarios se van a realizar pintando figuras en Java. No se van a usar herramientas diseñadas que permitan pintar figuras, sino que se van a desarrollar íntegramente por los programadores. Las clases serán rectángulos de ancho como el nombre de la clase. Las relaciones serán líneas rectas. El punto de anclaje entre una relación y su clase será el corte entre el lado de la clase y la recta de la relación.

Para familiarizarnos con OCL se nos proporcionó documentación, y se pidió plantear los objetivos de diagramas de clases para los diagramas de objetos.

Reunión 2, 17-10-05:

El programa desarrollado durante la semana permitía:

- Pintar clases y moverlas por la superficie de dibujo.
- Pintar relaciones y variarlas si se movían las clases de los extremos.
- Manejar opciones básicas gracias a un menú superior.

Todo esto se hacía en un sencillo entorno que contenía una superficie de dibujo y un panel con dos botones: clase y relación. El problema era que se trabajaba con atributos estáticos, por lo que siempre pintaba la misma clase ya que era más fácil para comenzar.

Los objetivos de esta semana serían:

- Pasar a un entorno dinámico para que el usuario pudiera poner clases y relaciones a su gusto.
- Crear un menú de edición de clases para poder cambiar atributos como el nombre.
- Crear un menú de edición de relaciones para cambiar sus atributos.
- Inserta atributos en una clase.
- Comenzar el guardado y carga en base de datos.

Tareas sobre las que se debería empezar a estudiar serían:

- Generar los diagramas de objetos de manera similar a los de clases.
- Estudiar el estándar XMI que usan otras herramientas como Poseidon de manera que podamos guardar y cargar este tipo de archivos, similar a XML. Al ser un estándar complejo hay que encontrar documentación y estudiarlo.

Se plantearon futuras funcionalidades de la aplicación como posibilidad de guardar en formato pdf, zoom de aumento y reducción, posibilidad de impresión de los diagramas…

24-10-05. No hay reunión.

Esta semana continúan los trabajos de la anterior debido a que plantear el diagrama de objetos implica hacer cambios en la base de la aplicación. La salida XMI está en desarrollo. La nueva versión presenta:

- Un menú para clases que permite cambiar su nombre, añadir atributos y eliminarlos.
- Un menú de relación que permite cambiar su nombre y roles, y ver las clases origen y destino.
- Trabajo con clic derecho del ratón que permite desplegar menú de opciones sobre los elementos del diagrama.
- Aparición del nombre de la relación en el diagrama de clases en el punto medio de dicha relación, y de los roles en los extremos.
- Mejoras de manejabilidad como la detección del doble clic sobre una clase o relación que despliega el menú de dicho elemento.
- Posibilidad de crear relaciones de una clase en sí misma, aspecto que no se contempló inicialmente.
- Ampliación del tapiz de dibujo gracias a unas barras de scroll lateral.

Se podían pulir algunos detalles, sobre todo en los menús, pues resultaban poco intuitivos y en el scroll lateral, puesto que las clases podían arrastrarse fuera del tapiz sin que las barras de scroll ampliaran el tamaño del mismo.

Con la base de datos y el diagrama de objetos pendientes se añaden nuevas tareas:

- Posibilidad de eliminar clases y relaciones.
- Mejora de menús agrupando opciones similares en el mismo menú.
- Creación de un botón cursor que simplifique la funcionalidad de los botones actuales. Sus dos usos serían:
  - · Mover: para arrastrar clases y acceder a atributos de clase y relación.
  - · Editar: para editar clases y relaciones a nivel diagrama.

Reunión 3, 4-11-05:

Esta semana se presentan las mejoras gráficas y la inclusión del diagrama de objetos:

- Eliminación de clases y relaciones. Cuando se elimina una clase se eliminan las relaciones que dependen de ella.
- Mejora de menús emulando aplicaciones comerciales: archivo, edición, ayuda…
- Inclusión de un tercer botón: botón cursor. Dicho botón no crea elementos en el diagrama sino que permite editarlos y moverlos.
- Inclusión de diagramas de objetos. Estos diagramas son similares a los de clases. Contienen tres botones: cursor, crear objeto y crear enlace. Los objetos se crean a partir de clases del diagrama de clases.

Se fija un objetivo principal: realizar una conexión de la parte desarrollada con el módulo ITP/OCL Tool, un verificador de restricciones OCL escrito en Maude. Con esto se permitiría trabajar a dicha herramienta con la información insertada en el entorno gráfico. Se comenzaría estudiando la sintaxis de los comandos.

Se plantean las mejoras a llevar a cabo:

- Posibilidad de guardar varias instancias de un diagrama de objetos.
- Posibilidad de realizar las relaciones en múltiples segmentos.
- Posibilidad de que las relaciones salgan del centro de la clase, no de los extremos, lo que dificultaba la visibilidad.

- Posibilidad de pintar puntas de flecha para distinguir visualmente origen y destino.

Con estas mejoras y el uso básico de la base de datos se podría dar por terminado el primer módulo y pasar a la conexión con ITP/OCL Tool. Después se debería ampliar con más funcionalidades.

Reunión 4, 18-11-05:

En esta reunión se presentan los siguientes objetivos cumplidos:

- Varias instancias de diagrama de clases y diagrama de objetos.
- Posibilidad de realizar las relaciones en múltiples segmentos.
- Las relaciones salen del centro de la clase. Con ello se consigue una notable mejora visual, y se permite que una clase tenga múltiples relaciones sin que estas se entorpezcan gráficamente.
- Las relaciones tienen punta de flecha para distinguir origen y destino. Además muestran en medio de la relación el nombre de la misma, y en los segmentos correspondientes los roles de origen y destino.
- El diagrama de objetos solo permite crear instancias de objetos cuyas clases hayan sido previamente creadas.

En esta sesión recibimos un breve tutorial sobre el módulo de unión de las dos herramientas del proyecto: el interfaz gráfico y el verificador de modelos ITP/OCL Tool. Esta interconexión se incluye dentro de los objetivos nada más terminar esta parte gráfica.

Además de este objetivo, se plantean otros referentes a la entrega actual y a la ampliación de funcionalidades:

- Incluir todos los tipos de relación disponibles en UML: agregación, dependencia, instanciación…
- Posibilidad de guardar en formato gráfico EPS: encapsulated post script.
- Crear diagramas de objetos asociados a un diagrama de clases, de manera que un diagrama de objetos solo vea las clases del diagrama de clases asociado y no la de todos los diagramas. Esto implica el trabajo a dos niveles puesto que los diagramas de clases pasan a ser independientes.
- Actualización de la base de datos a los cambios efectuados hasta el momento.
- Al igual que al crear el objeto se debe instanciar una clase ya creada, los enlaces entre objetos deben crearse a partir de relaciones del diagrama de clases previamente creadas. Además puede haber varias relaciones entre dos clases, no una sola como se había supuesto de partida.
- Uniformidad de apariencia en las ventanas de menú para que todas presenten el mismo aspecto.
- Posibilidad de ver gradualmente los segmentos creados a la hora de pintar una relación o enlace. La opción actual dejaba al usuario pinchar en los puntos, pero no mostraba la relación hasta el final.

Reunión 5, 30-11-05:

Esta semana se presentan todos los objetivos cumplidos y algunos adicionales:

- Nuevas relaciones UML: realización, generalización, instanciación…
- Creación de diagramas de objetos asociados a un único diagrama de clases. Los objetos solo pueden ver las clases de su diagrama padre. Para ello hay dos niveles de pestañas en la aplicación: Conjuntos y Diagramas. Los conjuntos son grupos de diagramas que incluyen un único diagrama de clases y varios de objetos. En el mismo conjunto, los diagramas de objetos instancian clases de su diagrama de clases. Se pueden tener varios conjuntos operativos, de modo que sus clases no se mezclen.
- Guardado en EPS del diagrama actual, conjunto actual o todos los conjuntos.
- Las relaciones entre clases no son únicas.
- Los enlaces entre objetos se eligen de entre las relaciones que tienen las clases instanciadas de los objetos origen y destino. Si no tienen relación entre si no puede haber enlace.
- Todos los menús se repasan para ser uniformes.
- Posibilidad de ver gradualmente los segmentos creados a la hora de pintar una relación, tanto para clases como para objetos.
- Zoom para alejar y acercar.
- Posibilidad de imprimir el diagrama actual.
- Scroll de pantalla plenamente operativo. Amplía el tapiz cuando un objeto o clase se sale de los límites.

Con estas mejoras va terminando la implementación de la parte gráfica y cobra más importancia la unión con ITP/OCL Tool. Ya se ha comenzado el proceso de interconexión. El primer paso ha sido poder arrancar Maude e ITP/OCL Tool desde la herramienta gráfica. Aunque no se ha realizado con éxito se espera terminarlo para poder empezar a mandar comandos.

Los objetivos planteados esta semana son:

- Adaptar la base de datos a los cambios realizados.
- No permitir cambiar datos en los enlaces entre objetos. Estos datos provienen del enlace entre clases por lo que son inalterables. Se debe eliminar el nombre del enlace ya que se consideran sin nombre.
- Los nombres de rol y relación deben adaptarse a la distancia entre clases u objetos ya que a veces son tapados por estos.

Reunión 6, 12-12-05:

Esta semana se presentan mejoras importantes para la aplicación:

- No permitir cambiar datos de un enlace entre objetos.
- Superposición de los roles y nombre de relación a las clases. Esto se ha conseguido separando el pintado de las relaciones y de sus textos en dos fases: primero la relación, luego los textos. Lo que se pinta lo último se superpone a lo anterior.
- Actualización parcial de la base de datos.

- Se ha realizado con éxito la conexión con ITP/OCL Tool a través de Maude. Para mandar los comandos se ha creado una jerarquía de trabajo de modo que se mandan los comandos de:
    - Cada diagrama de cada conjunto:
        - · Si es DC: se pasan clases y después relaciones.
            - Las clases implican pasar los atributos.
        - · Si es DO: se pasan objetos y después enlaces.
            - Los objetos implican pasar atributos y valores.

Respecto a estos nuevos comandos y la parte gráfica se deben pulir algunos problemas para la semana siguiente:

- Al enviar a Maude un comando sobre un atributo de clase de tipo enumerado se produce un fallo si la clase enumerada no se mandó a Maude previamente.
- La herramienta gráfica presenta acciones con doble clic y botón derecho que no funcionan con las dos opciones.
- Mejorar la sensibilidad de las relaciones y enlaces en respuesta al doble clic de ratón.

Reunión 7, 21-12-05:

Se presentan las mejoras de la semana pasada:

- Nuevo planteamiento de envío de comandos a Maude para diagramas de clase:
    - · Si es DC: se pasan clases enumeradas, clases normales y después relaciones. Las clases pasan sus atributos. Esto corrige el problema que se planteó
- Doble clic y botón derecho corregidos y con sensibilidad mejorada.

La herramienta visual está terminada a falta de pequeños retoques y la comunicación con Maude es completa y correcta. Para terminar quedan dos retos: en primer lugar hacer funcionar la base de datos en Linux. En segundo lugar realizar una nueva interacción con ITP/OCL Tool: el trabajo con invariantes.

Respecto a los invariantes se debe incluir un editor de invariantes en los diagramas de clases. Este editor confirmará que el invariante es sintácticamente correcto y lo mandará a Maude. Asimismo el sistema debe ser capaz de poder guardar los invariantes introducidos. Para los diagramas de objetos debe existir un botón de comprobación de invariantes de manera que se comprueben todos los invariantes insertados previamente en su diagrama de clases original.

Aparte del trabajo con Maude se presentan los siguientes retoques:

- Crear un fichero log de comandos que guarde las instrucciones mandadas a Maude.
- Las clases enumeradas no pueden ser instanciadas.
- Los conjuntos de pestañas deben tener nombres únicos.
- La generalización de clases lleva un comando Maude aparte: insert-subclass.
- Los roles de relación se inicializan con nombreclase@nombrerelacion.
- Se deben tratar las respuestas de Maude de manera que si se detecta un fallo no se pinte en la parte gráfica. Con esto pretendemos llevar la coherencia entre las dos partes.

- Se quieren asociar eventos de ratón a cuadros de escritura de manera que al pulsar Intro se haga la acción correspondiente sin necesidad de pulsar el botón.

Reunión 8, 9-01-06:

En el retorno de vacaciones se ha cumplido con todo el trabajo propuesto. Se han logrado los siguientes objetivos:

- Cada vez que se envían los comandos, se actualiza un fichero en el directorio raíz denominado *log.txt*.
- No se instancian clases enumeradas.
- Los nombres de conjuntos son los del diagrama de clases que contiene, al ser este único.
- Los roles tienen valores por defecto puesto que lo necesitan los comandos.
- Se asocian acciones de botones a cuadros de texto pulsando Intro.

Han quedado inconclusos dos objetivos por falta de información:

- La edición de invariantes debe ser corregida pues algunas órdenes pedidas no existen en el repertorio de órdenes. Se debe considerar dos tipos de invariantes ya que hay dos sintaxis distintas: si se pincha sobre una clase o si se pincha en vacío. La idea es por tanto: elegir el botón del Editor, pinchar en clase o en vacío, hacer aparecer el editor y poder escribir. Se mandará a Maude el comando correspondiente según donde se haya pinchado.
- La respuesta Maude no se analiza aún para decidir si actuar o no. De momento, al realizarse el paso a Maude de manera jerárquica se ha conseguido que todo funcione y ya se obtiene el resultado de este pase.

La solución para los invariantes es sencilla una vez se habla de nuevo sobre el repertorio de instrucciones. El trabajo con Maude ha sacado algunas ideas nuevas a la luz ya que en los comandos no pueden dejarse campos en blanco. Por ello se trabaja con valores por defecto cuando hablamos de objetos, y nombres y roles por defecto cuando hablamos de relaciones (lo que ya se trató antes de Navidad). Los atributos por defecto serán '0' para enteros, 'true' para booleanos, " " para cadenas y el primer valor enumerado para atributos enumerados.

Respecto a la base de datos ahora hay que conseguir que funcionen las tablas, puesto que parece que no se copian correctamente en el sistema. No permiten escribir. Como solución temporal las tablas se crean directamente a través de comandos SQL.

Para la próxima reunión se debe:

- Corregir el editor de invariantes para que funcione según lo visto anteriormente.
- Permitir guardar invariantes y visualizarlos.
- Corregir los valores iniciales para objetos, sobre todo en la parte referente a atributos enumerados.
- Permitir actuar según la respuesta de Maude.
- Tener los nuevos comandos de creación de tablas en SQL.

Reunión 9, 17-01-06:

Se presentan los objetivos cumplidos:

- Editor de invariantes operativo. Permite trabajar con los dos tipos de invariantes existentes. Realiza un análisis sintáctico previo al envío. Si el invariante es correcto se guarda en el sistema y se cierra el diagrama de clases pertinente.
- Existe un visor de invariantes para diagramas de clases que muestra los invariantes asociados y permite eliminar los que se deseen.
- Los comandos de SQL están terminados y funcionan correctamente en Linux.

El centro del trabajo se está desviando poco a poco hacia la base de datos puesto que es la única manera de que se dispone para guardar diagramas y poder recuperarlos posteriormente. Respecto a la base de datos se plantean algunas mejoras:

- Que se despliegue un menú a la hora de cargar fichero donde se muestren los ficheros guardados. Esto evita tener que poner el nombre del fichero como ocurría hasta ahora.
- Actualizarla para nuevos elementos incorporados como invariantes.
- Poder asociar un comentario breve a un diagrama de modo que se muestra a la hora de cargarlo.
- Arreglar alguna excepción que se produce al trabajar con objetos.

Para la herramienta gráfica ya había concluido la interacción con Maude al terminar correctamente con los invariantes. Era momento para pulir algunos detalles visuales y de funcionamiento:

- Incluir un menú preferencias para configurar las rutas de Maude e ITP/OCL Tool y el idioma (en mejoras futuras). Esto evita configurarlos en cada uso.
- Eliminar el botón instancia de los diagramas de clases al estar esto contemplado implícitamente en los diagramas de objetos.
- Posibilitar eliminar diagramas y limpiarlos.
- Cerrar Maude al salir de la aplicación.
- Permitir meter invariantes aún con el diagrama de clases cerrado.
- Corregir el editor de invariantes de manera que los invariantes introducidos se guarden directamente en el sistema. No hace falta un botón guardar como había.
- Los nombres de conjuntos pasan a ser los nombres de la clase, con lo que en el menú configuración desaparece esta posibilidad.

Estas mejoras están orientadas en parte a una división conceptual de lo que es la herramienta. Por un lado tenemos lo que es el proyecto en sí, es decir todos los conjuntos de diagramas abiertos con los que estamos trabajando. Para ello está el menú *File* que carga, guarda, limpia… Y en segundo lugar están los diagramas de cada conjunto con el que estamos trabajando. Para ello está el menú *Edit* que permite eliminar, borrar, crear nuevos diagramas, configurarlos…

Estando cerca de tener una versión de prueba plenamente funcional se plantean retos para posibles ampliaciones: textos bilingües, crear un instalador…

Ampliaciones más cercanas y posibles son: trabajar con clases de asociación, trabajar con multiplicidades, consultas…

Reunión 10, 26-01-06:

Todas las mejoras propuestas anteriormente se han cumplido. Además se ha realizado un lavado de cara a la aplicación. Se ha sustituido el color gris estándar de aplicaciones de Windows y su paleta de colores por una paleta personalizada para la aplicación. Del mismo modo se han sustituido las letras de los botones por iconos de modo que pueda ser comprensible en cualquier idioma.

Se incluyó un color que diferenciara el elemento seleccionado dentro de un diagrama en cada momento, ya fuera clase, objeto, relación o link. Dicho color es el naranja. Para terminar se ha añadido una segunda barra con botones de acceso directo a las funciones más habituales de modo que se puedan acceder sin tener que navegar por los menús.

En esta sesión se ha comparado la herramienta con otras similares desarrolladas por otros equipos y se ha visto que aunque el nivel gráfico puede estar retrasado, a nivel operativo el desarrollo está avanzado. Una de las cuestiones que no tratan otros programas es la herencia, tanto de atributos como de relaciones. Si bien es cierto que nuestra herramienta no lo cumplía, era el momento de realizarlo. Tras unos pequeños ajustes en los diagramas de clases y objetos se plantean los últimos cambios, relativos a las ampliaciones antes comentadas:

- La lista de relaciones disponibles entre dos clases a la hora de crear un link entre sus respectivos objetos pasaba ahora a ser un comando de ITP/OCL Tool. Con esto se evitaba realizar una búsqueda en la lógica del sistema que a la hora de tratar herencia se complicaba al tener que buscar a sucesivos padres. El nuevo comando devolvía directamente la lista de relaciones posibles tanto entre clases como entre padres implicados. Esto mejoraba el tiempo de tratamiento e introducía la herencia en un primer paso. El segundo paso fue la herencia de atributos, donde a través de otro comando se obtenían los atributos de una clase incluyendo los heredados de las clases padre correspondientes.

Con este paso de herencia se conseguían mejores resultados que otros equipos de desarrollo que no tenían en cuenta dicho factor.

Reunión 11, 02-03-06:

Para esta reunión se había integrado y verificado el paso más importante de la reunión anterior: la herencia. Junto con unos ajustes a nivel gráfico y de trabajo con Maude, se había cerrado una etapa importante, y se planteaban los nuevos y últimos objetivos:

- Guardar los invariantes en la base de datos de modo que se pudiera trabajar con ellos siempre que se deseara.
- Incluir multiplicidades en las relaciones, tanto simples como compuestas. Las multiplicidades simples son de la forma n ó * con n>=0. Las compuestas son de la forma n..* con n>=0 ó n..m con m>n y n>=0. Además estas multiplicidades debían adaptarse al nuevo comando de insertar relación.
- Incluir un editor de consultas para los diagramas de objetos de modo que se pudieran realizar consultas sobre ellos.
- Incluir clases de asociación en los diagramas de clases. Estas clases relacionan clases con relaciones, por lo que su inserción en Maude es distinta. A nivel gráfico se añade un nuevo procedimiento similar al de relación entre clases, pero

pinchando en segundo lugar sobre una relación. En nivel lógico se harán los cambios correspondientes, así como para reflejarlo en los diagramas de objetos.

Reunión 12, 20-04-06:

Los objetivos principales de la reunión anterior se han cumplido:

- Guardar y cargar invariantes en la base de datos.
- Añadir multiplicidades en las relaciones añadiendo el nuevo comando. La multiplicidad por defecto es *.
- Incluye editor de consultas que permite escribir y evaluar en la misma ventana.
- Incluye clases de asociación en el diagrama de clases.

Con estos retoques se está llegando a la primera versión plenamente operativa, cuya fecha aproximada para ser pública se ha estimado a finales de mayo.

ITP/OCL Tool sigue avanzando, y para llegar a su nivel se necesita el último paso de clases de asociación: objetos de clases de asociación. Para ello el usuario seleccionará una clase y después una relación. Si existe asociación a aplicar se realizará una instancia, si no se indicará lo contrario.

Reunión 13, 27-04-06:

Se ha llevado a cabo la última modificación en Visual ITP/OCL Tool incluyendo enlaces de asociación. Cuando la base de datos esté preparada para almacenar dichos cambios se tendrá la primera versión plenamente operativa y se comenzará con la realización de la memoria.

Quedan indicadas posibles mejoras para sucesivos grupos que continúen el trabajo: instalador, sistema multilingüe, inclusión de funciones en clases, compatibilidad con Windows…

Posteriores mejoras:

En trabajo posterior realizado en la herramienta se han logrado las siguientes mejoras:

- Carga y guardado en archivos XML y XMI para los diagramas.
- Carga y guardado de invariantes en archivos de texto.
- Retoques gráficos como mejora de colores, actualización de archivos eps, borrado de relaciones y enlaces de forma recursiva por si tienen asociados relaciones de asociación, mejoras de configuración, mejoras en la edición de invariantes…
- Revisión de la mejora en tiempos de ejecución. Se había observado cómo la ejecución de algunas acciones llevaba varios segundos de espera hasta la respuesta. Esto se debe a que el envío de comandos a Maude tras hacer un cambio se hace de todos los diagramas del proyecto. A más diagramas y más elementos más tiempo, lo que se nota considerablemente para diagramas grandes. Como los comandos de ITP/OCL Tool no permiten modificar un elemento, ni borrarlo, la única opción era la usada: mandarlo todo. Se han creado una nueva serie de comandos que permiten modificar sólo un elemento, lo que ha mejorado notablemente la eficiencia de muchas acciones. Se han tenido que modificar los métodos de paso a Maude de todos los elementos. Las medidas comparativas con

la herramienta USE se han repetido obteniendo mejores tiempos a nuestros favor. Se pueden consultar en el apartado de comparativas 3.3.

- Posibilidad de ejecución en Windows, lo que hacer a Visual ITP/OCL Tool un editor multiplataforma.

3.3 Comparativa de Visual ITP/OCL Tool frente a otros editores:

Visual ITP/OCL Tool presenta dos vertientes de trabajo distintas. En primer lugar, la edición de diagramas, y en segundo lugar, la validación de restricciones sobre diagramas de objetos. Existen numerosas herramientas, con mayor o menor difusión, que sirven para los mismos propósitos que Visual ITP/OCL Tool.

Rational Rose y Poseidon son los dos editores de diagramas más utilizados. Los dos son de pago, lo que impide su uso completo por parte del usuario. A nivel gráfico, los dos son superiores a Visual ITP/OCL Tool, y su editor de diagramas es mucho más completo. Hay que tener en cuenta que la versión de Visual ITP/OCL Tool presentada es la primera en desarrollo, por lo que no refleja la totalidad del proyecto. Visual ITP/OCL Tool es una herramienta de libre distribución lo que siempre es un punto a favor a la hora de utilizar software. Existen varias ventajas de Visual ITP/OCL frente a Rational Rose y Poseidon. En primer lugar la posibilidad de trabajar con invariantes y consultas, aspectos que no se contemplan en dichas herramientas. En segundo lugar, y de gran importancia, uso de herencia de atributos y relaciones entre clases usando relaciones de generalización. Esta herencia se refleja a la hora de crear objetos, los cuales heredan atributos y relaciones de los padres. La herencia no se contempla en las herramientas anteriores. Este hecho no es negativo, ya que dichos atributos se pueden editar en los objetos correspondientes, pero hace que los diagramas resultantes no reflejen la realidad del modelo dada por el diagrama de clases, lo que puede inducir a error a la hora de razonar. En tercer lugar y más importante, Visual ITP/OCL Tool presenta la posibilidad de validar restricciones OCL. Rational Rose y Poseidon no contemplan dicha posibilidad por lo que el usuario debería validar sobre el papel, lo que puede llevar a obtener deducciones erróneas.

En conclusión, aunque Visual ITP/OCL Tool es inferior gráficamente y en la edición de diagramas, cuenta con varios aspectos a su favor que Rational Rose y Poseidon no contemplan. Especialmente en la vertiente de validación de restricciones. Estas herramientas se limitan a editar, mientras que Visual ITP/OCL Tool edita y valida.

Estas mismas conclusiones se pueden aplicar sobre editores de libre distribución como EasyCase, UMLet o Violet, que presentan un interfaz muy similar a Visual ITP/OCL Tool, y muchas menos prestaciones que Rational Rose y Poseidon.

Existe una herramienta de libre distribución denominada USE, diseñada como editor y validador de modelos. Es la única herramienta existente comparable a Visual ITP/OCL Tool. El nivel gráfico es el mismo que Visual ITP/OCL Tool al estar desarrollados con Java y presentar un interfaz idéntico. USE es capaz de editar diagramas de clases, de objetos y de secuencia, lo que le sitúa a un nivel similar al de nuestra herramienta y por debajo de Rational Rose y Poseidon. Sin embargo, posee un motor de validación, para lo que como se ha dicho anteriormente basta con diagramas de clases y objetos.

En la vertiente de edición de diagramas, USE se encuentra en un nivel inferior a Visual ITP/OCL Tool. USE no permite añadir y eliminar elementos a sus diagramas dinámicamente utilizando un panel con opciones como usan los editores corrientes. Para crear un diagrama de clases en USE hay que editar un archivo de texto y diseñar el diagrama escribiendo comandos con una sintaxis propia. Este hecho tiene dos consecuencias negativas: en primer lugar obliga al usuario a aprender y comprender la sintaxis que necesita, y en segundo lugar ofrece diagramas estáticos que una vez cargados no se pueden modificar. La única manera de modificar es editar el archivo, modificarlo y volver a cargarlo. Esto es un planteamiento bastante tedioso puesto que es mucho más sencillo poder editar gráficamente diagramas y modificarlos según desee el usuario. El mismo problema existe para los diagramas de secuencia.

Este hecho de tener una sintaxis propia y generar diagramas estáticos tiene influencia a la hora de trabajar con intercambio de archivos. USE no permite cargar ni salvar en archivos XMI o XML, lo que obliga a los usuarios a rehacer sus diagramas completamente si desean editarlos en otra herramienta, o convertirlos a la sintaxis de USE si es el caso inverso. Este inconveniente no ocurre en Visual ITP/OCL Tool que permite cargar y salvar en XML y XMI, y hacer uso de una base de datos y archivos Eps.

En la vertiente de validación de restricciones sobre diagramas de objetos, USE y Visual ITP/OCL Tool presentan dos motores de validación distintos. El motor de USE permite la edición y comprobación de precondiciones y postcondiciones sobre operaciones, aspecto no contemplado en Visual ITP/OCL Tool. La comparación de dichos motores se va a realizar a través de una prueba sobre dos diagramas de objetos. Los diagramas son variantes del diagrama de clases TRAINWAGON comentado en la introducción. TRAINWAGON-10x25 presenta 10 trenes con 25 vagones cada uno, esto es 250 vagones. TRAINWAGON-10x100 presenta 10 trenes con 100 vagones cada uno, esto es 1000 vagones. Todos los vagones en ambos diagramas están correctamente enlazados, sin formar ciclos, y presentando vagón sucesor y predecesor excepto para el primer y último vagón de cada convoy.

Sobre los dos diagramas de objetos verificaremos 4 restricciones con el significado propio de su nombre:

- alMenosUnVagon.
- pertenecenAlMismoTren.
- mismoNumeroDeVagones.
- noDeFormaCiclica (el mismo notInCyclicWay de la introducción).

Los resultados obtenidos fueron los siguientes:

	TRAINWAGON -10x25		TRAINWAGON-10x100	
	USE	ITP/OCL	USE	ITP/OCL
alMenosUnVagon	0.055s	0.076s	0.034s	0.116s
pertenecenAlMismoTren	0.207s	0.076s	0.970s	0.780s
mismoNumeroDeVagones	0.202s	0.120s	0.241s	0.936s
noDeFormaCiclica	4.793s	1.704s	280.209s	26.43s

Como se puede observar, la primera restricción presenta mejor tiempo de ejecución en USE que en ITP/OCL Tool. Esta diferencia es imperceptible para el usuario debido al rango de tiempo empleado en la ejecución.

Las dos siguientes restricciones presentan por los general peores tiempos en USE, exceptuando mismoNumeroDeVagones para TRAINWAGON-10x100.

Para TRAINWAGON-10x25 USE tarda entre 5 y 20 centésimas de segundo, e ITP/OCL Tool entre 7 y 12 lo que le otorga cierta ventaja. Para TRAINWAGON-10x100 USE tarda entre 3 y 97 centésimas de segundo, e ITP/OCL Tool entre 11 y 93. En este caso la diferencia es mayor que para TRAINWAGON-10x25 ya que el número de objetos en el diagrama es mayor. USE obtiene en este caso una mínima ventaja frente a ITP/OCL Tool.

Estas tres primeras restricciones son sencillas y no requieren de gran cantidad de cálculos. Considérese ahora la cuarta restricción noDeFormaCilica. Dicha restricción requiere muchos cálculos, y el número de objetos del diagrama es un factor influyente en el tiempo de cálculo. Para TRAINWAGON-10x25 USE tarda alrededor de 5 segundos, e ITP/OCL Tool menos de 2 segundos. Esta diferencia es considerable comparada con los resultados de las 3 primeras restricciones. En TRAINWAGON-10x100 USE tarda casi 5 minutos en devolver el resultado, mientras que ITP/OCL Tool tarda poco más de 25 segundos. Esto muestra que restricciones complejas afectan enormemente a USE, y que dicho factor se agrava a medida que aumenta el número de objetos en el diagrama.

En conclusión Visual ITP/OCL Tool es una herramienta con posibilidades de edición de diagramas, carga y almacenamiento superiores a las ofrecidas por USE. En el aspecto de validación de restricciones, USE presenta mejor eficiencia para algunas restricciones con poco número de cálculos, pero a medida que crecen los cálculos y el número de objetos Visual ITP/OCL Tool presenta una eficiencia muy superior. USE permite validar precondiciones y postcondiciones sobre operaciones y crear diagramas de secuencia estáticos. Estos hechos pueden llevar a pensar que una futura versión de Visual ITP/OCL Tool con mejoras y posibilidades que no se han podido implementar aún, puede ser la herramienta de uso general para editar y validar restricciones en vez de USE, gracias a las ventajas y mejoras notables que ofrece sobre la misma.